САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

**ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ – ПРОЦЕССОВ УПРАВЛЕНИЯ**

**КАФЕДРА ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ**

**Ефремов Александр Владиславович**

**Дипломная работа**

# Разработка программного продукта для поиска аналогов документа с помощью поисковой системы Яндекс

Заведующий кафедрой,
кандидат физ.-мат. наук,
профессор Сергеев С. Л.

Научный руководитель,
ст. преподаватель Малинина М. А

Рецензент,
к.т.н. Анфиногенов Сергей Олегович

Санкт-Петербург

2016

# Содержание

Введение ..... 3

Постановка задачи ..... 4

Обзор существующих решений ..... 5

Глава 1. Описание работы системы. Основные проблемы и решения.... 8

Глава 2. Алгоритмы сбора обучающей коллекции и обработки документов. ..... 18

2.1. Поиск по алгоритму Бойера - Мура. ..... 18

2.2. Скачивание документов. ..... 21

2.3. Декодирование символов из UTF8 в CP1251 и обратно……22

2.4. Выделение предложений……...23

2.5. Лемматизация……27

2.6. Сравнение строк с помощью хэш таблиц……27

2.7. Кодирование документов……..31

2.8. Управление коллекцией……32

2.9. Настройки ..... 34

Глава 3. Алгоритмы подсчёта частот и поиска аналогов документа. ... 36

3.1. Подсчёт частотного словаря пар слов. ..... 36

3.2. Поиск документов аналогов. ..... 38

Глава 4. Результаты……41

Глава 5. Выводы ..... 45

Заключение ..... 46

Список литературы ..... 47

Приложение ..... 49

# Введение.

В жизни современного человека всё большую ценность приобретает владение информацией. Обладая знаниями о чужих ошибках, человек может избежать аналогичных неудач. При наличии необходимой информации можно намного быстрее и, зачастую, эффективнее решать типичные задачи.

Благодаря всемирной сети Интернет, можно найти практически любые данные по интересующей теме. За годы своего существования человечество накопило огромное количество информации. Человеку не хватит всей жизни, чтобы освоить хотя бы тысячную долю всех доступных ему знаний. Данная ситуация не столь радужна, как кажется. Как правило, своевременно и оперативно получить необходимую информацию практически невозможно. Это обусловлено большими объемами доступной в Интернете информации, не всегда корректно структурированной.

Для облегчения задачи разработаны специальные поисковые системы, к примеру, Яндекс или Google, которые могут по запросу из нескольких слов выводить ссылки на релевантные документы или web-сайты. Однако, результат зависит от корректности формулировки запроса. Периодически возникают ситуации, в которых по существующему документу необходимо получить дополнительную информацию, уточняющего или дополняющего характера. Но пользователю достаточно сложно правильно составить запрос для получения нужных ему документов. В рамках данной работы выдвигается гипотеза о возможности упрощения задачи корректной формулировки запроса с помощью автоматического составления списка слов для построения релевантного запроса.

# Постановка задачи.

Целью данной работы является разработка программного комплекса, позволяющего по исходному документу составить список слов, которые пользователь сможет использовать для поиска аналогов документа и нахождения необходимой ему информации.

Необходимо разработать программный продукт, обладающий следующими возможностями:

- Автоматическая сборка обучающей коллекции документов из Интернет
- Разделение текстов на предложения, очистка от HTML кода и всего что не является предложением (ссылки, списки таблицы и т. д.).
- Лемматизация слов – приведение слов к так называемой нормальной форме: именительный падеж и единственное число для существительных, инфинитив для глаголов и т. д.
- Составление нумерованного словаря и кодирование документов по нему
- Подсчёт частоты встречи комбинаций пар слов в пределах предложения во всем наборе документов обучающей коллекции.
- Составление списка слов исходного документа, рассчёт их релевантности для использования в запросе к поисковой системе, сортировка в порядке снижающейся релевантности и их вывод, вместе с данными позволяющими оценить полученные значения релевантности.

# Обзор существующих решений.

## Нечёткая кластеризация.

Кластеризация — это задача разбиения множества объектов на группы, называемые кластерами. Внутри каждой группы должны оказаться похожие объекты, а объекты разных групп должны отличаться друг от друга как можно больше. Особенность нечеткой кластеризации состоит в том, что кластеры являются нечеткими множествами, и каждый элемент принадлежит различным кластерам с различной степенью принадлежности [1].

Существует ряд алгоритмов для нечёткой кластеризации и все они сводятся к вычислению центров заранее заданного количества кластеров по какой-либо функции от матрицы разбиения, значения элементов которой означают принадлежность элемента множества определённому кластеру. Далее матрица разбиения обновляется, с учетом вычисленных центров и проверяется, удовлетворяет ли она поставленному условию. Если нет, то алгоритм повторяется с обновлённой матрицей до тех пор, пока не будет выполнено условие.

Для применения нечёткой кластеризации к поиску аналогов документа, первоначально необходимо выполнить кластеризацию базы документов, в которой предполагается производить поиск. Для документа, выступающего в качестве запроса, необходимо вычислить меру близости к кластерам и из самых близких составить список документов кандидатов. Затем кандидаты сравниваются с запросом, а в результате выводятся лишь те документы, в которых набор слов достаточно близок набору слов документа, использованного в качестве запроса.

Достоинства данного метода в значительном сокращении времени рассчёта при поиске за счёт того, что документ сравнивается с центрами кластеров и впоследствии только с теми документами, которые принадлежат этим кластерам. Для этого необходима предварительная обработка

документов базы, связанная с большими вычислительными затратами Кроме того, недостатком является необходимость предварительного задания количества кластеров в коллекции. Предполагается, что кластер включает в себя документы по одной теме. Если состав базы не известен или планируется её расширение, то верно определить количество кластеров достаточно сложно.

В данной работе ставится задача поиска аналогов документа в Интернете, а кластеризовать весь Интернет задача практически невыполнимая. Следовательно, указанный метод в рассматриваемом случае не применим.

## Алгоритм LSA.

Латентно-семантический анализ (LSA) — это метод обработки информации на естественном языке, анализирующий взаимосвязь между коллекцией документов и терминами в них встречающимися, сопоставляющий некоторые факторы (тематики) всем документам и терминам [2].

Если составить матрицу слова-документы $\{A_{ij}\}$, элементы которой равны частоте встречи i-того слова в j-том документе, то можно определить близость документов, как расстояние соответствующих им векторов. LSA позволяет построить с помощью этой матрицы две других: слова-темы и темы документы, а так же значительно сократить объём данных.

Для этого из матрицы слово-документ исключаются слова, встреченные лишь в одном документе, а затем применяется сингулярное разложение $A=USV^T$ где U и V ортогональные матрицы, а S - диагональная, значения на диагонали которой называются сингулярными числами. Далее по заранее заданному критерию выбираются наибольшие сингулярные числа. Остальные удаляются вместе с соответствующими им столбцами U и $V^T$, что не только сокращает размерность матриц, но и позволяет лучше выделить

принадлежность слов и документов к темам [3].

Чтобы по документу найти его аналоги, для каждого документа необходимо составить вектор частоты встречи всех слов и по нему, с помощью матрицы слово-тема определить самые близкие темы. Похожими будут считаться документы, отвечающие этим темам в матрице тема-документ.

К достоинствам данного метода можно отнести: снижение размерности данных, выявление не очевидной семантической связи слов с документами и возможность находить близкие по теме документы, не содержащие слов из документа образца. Недостатки аналогичны недостаткам нечёткой кластеризации: необходимость предварительной обработки коллекции, по которой производится поиск, плохая расширяемость и высокая сложность вычислений.

Для поиска аналогов документа в Интернете этот метод так же не подходит.

# Глава 1. Описание работы системы. Основные проблемы и решения.

В основе работы лежит гипотеза: в каждом предложении содержится одна законченная мысль. Предполагается, что парные комбинации слов, используемые в предложениях одной тематики, будут схожи. Это дает возможность сопоставить каждую тему с некоторым множеством комбинаций пар слов, наиболее часто встречающихся при изложении данной темы. Из таких определяющих множеств исключаются общие слова, часто встречающиеся в предложениях по любой теме (предлоги, часто используемые глаголы, прилагательные, вводные слова и т.д.). Оставшийся набор слов является специфическим для рассматриваемой темы. Ввиду сложности построения данных множеств, а также отсутствия необходимости их применения при решении задачи, такие наборы не строятся. В работе лишь предполагается, что они существуют. Для решения задачи достаточно данных о частоте встречи пар слов в предложениях.

Чтобы составить статистику частоты встречи комбинаций из пар слов, нужна обучающая коллекция документов, состоящая из множества текстов разной тематики с большим количеством документов по каждой теме. Так как частота встречи пар слов, описывающих тему, должна быть достаточно высокой. Изначально планировалось собирать коллекцию из статей Википедии и дополнительно по 10 документов найденных по запросу, состоящему из названия статьи. Увы, это потребовало бы весьма продолжительного времени на сбор. По данным на 7 января 2016 года в русскоязычной Википедии содержится 1 280 000 статей [4]. Таким образом, в общем, должно было получиться чуть более 12 миллионов документов. Если бы скачивание и обработка одного документа занимала бы всего секунду, то общее время выполнения равнялось бы 148 дням. Поэтому пришлось ограничиться нижеприведенными 4 областями и составленными

для них вручную  темами для запросов, по 50 документов на каждый запрос.

Далее приведен принцип, по которому составлялись запросы и примеры запросов с указанием релевантности. Релевантность вычислялась как отношение числа документов, содержание которых соответствует теме запроса с искомым смыслом, к количеству проверяемых документов. Выбор производится случайным образом. Проверялись 10 документов.

1). "Наука" - запросами служили научные термины и названия (400 запросов). Примеры:

- Предел 0.8
- Экспонента 0,2
- Катион 0,8
- Инерция 0,8
- Астрономия 0,5

2). "Экономика" - в качестве запросов использованы названия статей по экономике с сайта cyberleninka.ru (200 запросов). Примеры:

- Роль и место малых предприятий в современных экономических условиях 0,9
- Экономическая эффективность рекламы 1,0
- Внутренний валовый продукт 0,9
- Выбор ставки дисконтирования 1,0
- Проблемы и перспективы особых экономических зон 0,8

3). "История" - запросы взяты из книги "1000 самых важных исторических событий" (300 запросов). Примеры:

- Основание Рима 0,8
- Ледовое побоище 1,0
- Приход к власти Петра первого 0,9
- Трафальгарская битва 1,0
- Атомная бомбардировка Хиросимы и Нагасаки 0,8

4). "Политика" — В этой области запросы - названия новостей о политике из нескольких новостных лент (250 запросов). Примеры:

- Дональд Туск поздравил Дуду с победой на выборах президента Польши 0,8
- В Астане обсудили повестку предстоящего заседания совета ЕЭК 1,0
- Кабинет министров назначил служебное расследование по начальнику госэкоинспекции 0,5
- Медведев объявил выговор главе росимущества Ольге Дергуновой 0,9
- Контрольно-надзорную деятельность в России систематизируют 0,8

Средняя релевантность  всех документов по вышеприведённым данным 81%.

Бытует мнение, что поисковая система Google выдаёт более релевантные документы по запросу, поэтому первоначально планировалось использование данной системы для сбора коллекции. Был написан модуль программы, который по названию темы формировал ссылку для запроса, скачивал страницу и по признакам форматирования определял ссылки на документы, которые выдавал Google. Но спустя некоторое время Google

изменил принцип выдачи результатов. Сначала скачивалась пустая страница, а лишь после этого скрипт догружал список результатов. Из-за этого пришлось искать другую поисковую систему, которая позволяет получать результаты сразу на html странице, скачанной по запросу. Решено было переключиться на поисковую систему Яндекс, так как релевантность этой системы также достаточно хороша. Кроме тго, в данной поисковой системе удовлетворяются и другие необходимые для работы требования. Алгоритм работы модуля остался практически таким же.

При написании модулей скачивания страниц, выделения предложений, лемматизации и кодирования, программа тестировалась на небольшом количестве тем 5-10, что не вызывало проблем. Но как только был запущен сбор коллекции, IP адрес компьютера, на котором вёлся сбор, был добавлен в чёрный список Яндекса за автоматические запросы. Большое количество автоматических запросов от разных программ, используемых для сбора информации, приводит к серьёзной нагрузке на сервера поисковой системы, поэтому используются специальные алгоритмы определения, действительно ли человек работает с системой. Эти алгоритмы основаны на проверке тем и частоте запросов. Так как темы изменять было не желательно, а введение в программу ожидания между запросами не помогло, пришлось обратиться в службу поддержки Яндекса с просьбой разрешить автоматические запросы для сбора коллекции и выполнения данной работы.

Оказалось, что Яндекс имеет специальный сервис для автоматических запросов «Яндекс XML», но для того чтобы им пользоваться необходимо обладать сайтом для которого и предполагается использовать автоматические запросы. Удалось договориться со специалистом службы поддержки, что для данного исследования  выделят 5000 автоматических запросов.

В результирующей версии программы решено было оставить оба варианта получения ссылок, которые можно переключить с помощью

соответствующей настройки. В первом никаких дополнительных сведений не требуется, но можно собрать только очень небольшую коллекцию, во втором нужен аккаунт Яндекс XML с разрешением на определённое количество автоматических запросов, но размер коллекции неограничен.

Модуль скачивания страниц имеет возможность не скачивать документы если они находятся на сервере из списка исключений и запоминать ссылки по которым уже были скачаны документы, чтобы они не попали в коллекцию более одного раза.

Первым этапом обработки полученных документов является выделение предложений и очистка от HTML кода и всего, что не является предложением. Строка символов считается предложением, если начинается с заглавной буквы или тире и заканчивается на точку, вопросительный или восклицательный знаки. А также если расположена между тегами контейнерами и не прерывается тегами, наличие которых свидетельствует о переходе на следующий абзац.

Для того, чтобы работать с документом необходимо сначала определить его кодировку, и это вызывает определённые трудности. Дело в том что кодировка в HTML документе указывается в теге <meta>, который, в принципе, не обязателен. Записи о кодировке могут находиться в разных частях страницы, причём, несколько раз. Бывает и такое, что автор указывает кодировку неправильно, но самую большую сложность представляют документы, написанные сразу в двух кодировках. Алгоритм программы достаточно точно может определить кодировку документа, но если она не поддерживается или по каким-либо причинам не определена, то документ исключается из коллекции.

В русскоязычном Интернете наиболее распространены кодировки UTF8 и CP1251, он же WINDOWS 1251. Намного реже встречаются Unicode и KOI8R. Так как данное исследование проводится для текстов на русском языке, основной кодировкой выбрана CP1251, потому что с ней достаточно

просто работать, она требует всего один байт на запись символа и содержит все необходимые символы. Поддерживается перекодирование из UTF8. Остальные кодировки не используются ввиду их низкой распространённости.

Выделение предложений в документе затруднено свободным синтаксисом HTML. Иногда сложно предположить, что форматирование страницы может быть написано настолько не логично. К примеру, автор в начале текста использует тег <p>, а далее пользуется тегами <br> для переходов на следующий абзац. Непонятно, что мешало ему использовать тег <p>. Также нужно учитывать, что точка может означать сокращение, а не конец предложения. Обе эти проблемы решаются в алгоритме выделения предложений, хотя и не всегда верно, но результаты удовлетворяют необходимую точность.

Следующий этап обработки – лемматизация. В первую очередь необходимо разделить текст на токены. Токены — последовательности символов в лексическом анализе в информатике, соответствующие лексеме – слову, как абстрактной единице морфологического анализа. Для этого нужно отделить от слов знаки препинания. При этом, не отделяя знаки внутри слова, чтобы не нарушить целостность токена. Задача лемматизации достаточно сложна и хорошо известна в компьютерном анализе текста на естественном языке. Есть множество программ и библиотек для её решения. В данной работе решено было использовать одно из готовых решений для лемматизации.

Первый модуль лемматизации был написан для работы с программой TreeTagger [7]. Эта программа для определения частей речи, а лемматизация – это побочный результат её работы. На этапе разработки данного модуля казалось, что данные о том, какой частью речи является слово, могут быть полезны. Впоследствии стало понятно, что они не нужны, а также выяснилось, что TreeTagger не лемматизирует слова, которых нет в его словаре. Из-за чего размер словаря собранного по документам коллекции

оказался слишком велик. Его размер был чуть более миллиона слов. А так как впоследствии необходимо было подсчитать частоты встреч их пар в предложении, то на хранение данных потребовалось около 2 терабайт места на жёстком диске, что было неприемлемо.

Вторая версия модуля лемматизации использует библиотеку для лемматизации из SDK Грамматического Словаря [14]. В ней реализован алгоритм лемматизации, пытающийся предсказать лексему даже для неизвестных слов, используя характеристические функции. Она способна разрешать неоднозначности. Например, в предложении: «Мы ели суп, а вдоль аллеи стояли раскидистые ели.» - правильно определяется первое «ели» - глагол с инфинитивом «есть», а второе «ели» - существительное «ель», с помощью данных об окружающих их словах. По словам автора данной библиотеки, точность лемматизации равна 99% [5]. С помощью нового модуля лемматизации удалось сократить размер словаря до 600 000 слов. Его единственный недостаток-более долгое выполнение, сравнительно с TreeTagger-ом.

Компьютерная работа со словами естественного языка осложнена тем, что слова имеют переменную длину. Также для эффективного хранения данных в памяти, необходимо по значению слова быстро определить в какой ячейке памяти хранится соответствующее ему значение. Обычно, чтобы эффективнее использовать память, эти значения располагаются последовательно с постоянным шагом, необходимым для перехода от одной ячейки со значением к следующей. С этой точки зрения более логично закодировать слова числами, соответствующими их номеру в словаре составленном программой. Последний этап обработки документа – это кодирование. Помимо своей основной функции модуль кодирования может определять плагиат, для того чтобы при подсчёте частот они не завышали частоты пар слов из часто копируемого предложения. При кодировании ведётся подсчёт частоты встречи слова в коллекции для сокращения словаря

частоты пар слов путём исключения из него слов написанных с ошибками или настолько редко встречающихся, что данные по ним не дадут никакого результата.

Когда коллекция собрана и данные обработаны соответствующим образом можно приступать к подсчёту частот пар слов. Выполнение данной задачи весьма трудозатратно с точки зрения ресурсов компьютера. Таблица частот имеет настолько большой объём, что её невозможно целиком поместить в оперативную память. Приходится постоянно выводить данные по паре байт на жёсткий диск. По статистике [6] средняя длина предложения в русском языке 10 слов. Для записи частот пар слов среднестатистического предложения потребуется 100 операций чтения и 100 операций записи. Если в коллекции будет содержаться 5 000 000 предложений, то потребуется 1 000 000 000 операций ввода-вывода, для вычисления. Обычные жёсткие диски могут выполнить только 120 операций случайного ввода-вывода в секунду. Это связано со скоростью вращения считывающей головки, обычно 7200 оборотов в минуту. Таким образом, потребуется примерно 96 дней для рассчётов на одном жёстком диске. Благодаря дискам SSD, позволяющим осуществлять от 20 000 до 80 000 операций случайного ввода-вывода в секунду, необходимое время выполнения уменьшится на них в 200 раз.

В процессе выполнения данной работы была собрана обучающая коллекция из 56 700 документов, по 1134 темам -в среднем, по 50 документов на тему. Было использовано 71916 ссылок. В коллекции содержится 4 681 693 предложения, а словарь насчитывает 643 350 слов.

| **Частота T встречи слова** | **Число слов с данной частотой** |
|---|---|
| 1 | 637853 |
| 2 | 294987 |
| 3 | 213009 |

| | |
|---|---|
| 4 | 174487 |
| 5 | 151062 |
| 10 | 99798 |
| 50 | 41377 |
| 100 | 28485 |
| 500 | 11145 |
| 1000 | 7066 |
| 5000 | 2121 |
| 10000 | 1142 |
| 25000 | 385 |
| 50000 | 160 |
| 65535 (максимальная частота) | 111 |

Таблица 1.1. Частота встречи слов в коллекции.

По таблице 1 видно, что количество слов с частотой T достаточно сильно снижается при переходе с 1 на 2 и с 2 на 3, поэтому при подсчёте частот пар слов используются слова встретившиеся в коллекции хотя бы 3 раза. Память, потребовавшаяся для хранения таблицы частот равна 84,5 гигабайт.

Получив необходимые данные, можно приступать к поиску аналогов документа. Документ принимается в виде текста в кодировке CP1251 и проходит все стандартные этапы обработки: выделение предложений, лемматизацию и кодирование. При кодировании новые слова добавляются в коллекцию с частотой 0. Для получения релевантности слова запросу, используются четыре функции:

**Частота в исходном документе** – равна соответственно частоте встречи слова в документе, по которому необходимо найти аналоги.

**Известность в коллекции** – насколько часто встречалось слово при подсчёте частоты пар слов.

**Уникальность** – это значение обратно-пропорциолнально количеству слов, с которыми встречалось данное слово в предложении.

**Принадлежность исходному документу** – значение данной функции выше, если слово часто встречалось в одном предложении со словами из исходного документа, которые имеют высокое значение уникальности.

Значения этих функций используются в функции релевантности, после чего слова сортируются в порядке понижения значений релевантности. Результаты выводятся в файл, в котором пользователь сможет найти строку из сорока слов наиболее рекомендуемых для запроса, и список всех слов документа со значениями всех вышеописанных функций, для оценки работы программы.

В заключении необходимо отметить, что время, затраченное программой на сбор и подсчёт всех данных составило около 7 суток. Поэтому одной из самых главных проблем при разработке данного программного комплекса является огромный объём данных, которые вручную сложно проверить, а так же очень длительное время выполнения сбора и обработки обучающей коллекции.

# Глава 2. Алгоритмы сбора обучающей коллекции и обработки документов.

## 2.1. Поиск по алгоритму Бойера - Мура.

Алгоритм Бойера – Мура – один из алгоритмов поиска подстроки в строке. Широко используется в программе, к примеру, для поиска ссылок на страницах, полученных от поисковой системы, или для определения кодировки текста. В рамках данной работы была выполнена собственная реализация данного алгоритма.

Данный алгоритм был выбран потому, что считается наиболее быстрым среди алгоритмов общего назначения [8]. Преимущество этого алгоритма в том, что в результате некоторого количества предварительных вычислений  по подстроке, появляется возможность сравнивать с исходным текстом не всю подстроку — часть проверок пропускаются как заведомо не дающие результата.

Предварительно необходимо составить таблицу стоп символов и таблицу суффиксов.

Длина таблицы стоп символов равна количеству символов в алфавите. Так как основной кодировкой программы является CP1251, в которой содержится 256 символов, то размер таблицы также равен 256. Введем некоторые обозначения: длина подстроки - SL, позиция символа в подстроке - $S_n$, номер ячейки таблицы — $Tp_n$. Таблица стоп символов составляется по следующему алгоритму:

1. Вся таблица заполняется значением равным длине подстроки.
2. Ячейки таблицы с номерами равными номеру символа в CP1251 для всех символов подстроки начиная с первого и заканчивая предпоследним приравниваются к $TP_n = SL - S_n - 1$. Таким образом если в подстроке есть одинаковые символы, то в соответствующих

им ячейках будет значение при подсчёте которого использована позиция в которой он встречался последний раз.

Суффиксами подстроки называются последовательности из N символов совпадающие с последними N символами слова. Обычно для рассчёта таблицы суффиксов используются суффикс и префикс функции [8], что требует минимум четыре прохода по подстроке:

1). Функция суффиксов от подстроки.

2). Изменение порядка символов в подстроке на обратный.

3). Функция суффиксов повторяется.

4). Сравниваются значения обеих функций и определяются значения таблицы суффиксов.

Такой алгоритм вычисления в общем случае выполняется быстрее и его время выполнения зависит только от длины подстроки. Но так как в данной работе поиск проводится в текстах на естественном языке, то используется алгоритм собственной реализации с переменным временем выполнения, но более быстрый для таких текстов. Его время соответствует одному –двум проходам по подстроке. Обозначим максимальный суффикс MS, позиция символа в подстроке $S_n$ , длина подстроки SL, номер ячейки таблицы суффиксов $Ts_n$. Алгоритм представляет собой последовательность шагов:

1). MS=0

2) Для каждого Sn начиная с предпоследнего определяется, какой суффикс следует за ним, путём сравнения символов в позициях $S_n$ и $S_{SL}$ , $S_{n-1}$ и $S_{SL-1}$ и так далее.

- Если символы не совпадают, то осуществляется переход к следующей позиции, за которой ищется суффикс.

- Если символы совпадают и длина полученного суффикса больше значения максимального суффикса, то MS увеличивается на единицу, а в ячейку таблицы суффиксов с номером MS добавляется значение $TS_{MS} = SL - S_n$, где $S_n$ – позиция в подстроке с которой

начался поиск суффикса.

- Если поиск суффикса достигает первого символа слова и этот символ входит в суффикс, то оставшаяся часть таблицы суффиксов заполняется $TS_{MS} = SL - S_n$, где $S_n$ позиция с которой начался поиск суффикса.

3). Оставшиеся ячейки таблицы суффиксов равны длине подстроки $TS_{MS} = SL$.

Результат работы данного алгоритма - таблица размером SL-1, по которой можно определить за какой, пронумерованной с конца, позицией подстроки следует суффикс длиной $TS_{MS}$.

Алгоритм поиска основан на трёх идеях:

1). Поиск ведётся слева направо, сравнение подстроки справа налево. То есть последний символ подстроки сравнивается с позицией поиска в строке. Если они совпадают, то сравниваются символы, расположенные слева. Если нет, позиция сравнения в строке продвигается вправо.

2) Эвристика стоп-символа. Если последний символ подстроки не совпал с символом в позиции сравнения строки, то осуществляется перенос позиции сравнения на длину равную значению, содержащемуся в таблице стоп-символов, соответствующему не совпавшему символу.

3). Эвристика совпавшего суффикса. Если при сравнении совпала только часть подстроки, то позиция сравнения строки сдвигается на значение из таблицы суффиксов, так чтобы при существовании суффикса, позиции его символов совпали с его позициями в подстроке.

В модуле программы отвечающем за поиск реализована возможность сравнения подстрок как с учётом регистра, так и без него. Это достигается благодаря преобразования заглавных букв в прописные в подстроке перед подсчётами и в строке при сравнении.

## 2.2 Скачивание документов.

В данной работе разработаны два варианта скачивания страниц. Их алгоритмы очень похожи, поэтому далее рассматривается только алгоритм для работы с Яндекс XML.

Непосредственно скачивание HTML страницы по протоколу HTTP выполняет библиотека WinINet [9]. Для этого, методам библиотеки необходимо передать имя сервера и запрос к нему. Алгоритм скачивания страниц по определённому запросу к поисковой системе следующий:

1). В модуль программы, отвечающий за скачивание, передаётся название папки, в которой будут храниться документы и тема запроса.

2). Составляется запрос к серверу «www.yandex.ru» по принципу:

«/search/xml?query=”theme”&page=”N”&”yandex_xml_options”»

где “theme” – тема запроса, “N” – номер страницы результатов поиска, а “yandex_xml_options” содержит данные аккаунта для доступа к сервису, а также некоторые дополнительные настройки, описанные в документации[10]. Количество результатов изменяется от 10 до 100 на одной странице в зависимости от настроек Яндекс XML. В корневой папке создаётся папка «index» и в неё сохраняется полученная страница номер N под именем N.xml.

3). На скачанной с Яндекса странице ищется подстрока «<url>http». Такого вида запись предваряет ссылки на документы, найденные поисковой системой. Данный способ нахождения нестабилен. Если Яндекс изменит формат страницы, программа сработает с ошибкой. Но такое происходит не часто. Многие сайты в последнее время стали переходить на защищённое соединение по протоколу HTTPS. При этом также остается возможность соединения по протоколу HTTP, поэтому если после записи «<url>http» находится «s», то она пропускается вместе с символами «:\\». Всё что следует после, вплоть до знака «<» является ссылкой на найденный документ. Так как при повторном поиске сравнение начинается с места, в котором

закончилось в прошлый раз, то следующий поиск находит следующую ссылку. Если ссылки на странице заканчиваются, выполняется пункт 2 для следующей страницы с результатами поисковой системы.

4). Полученная ссылка отправляется в модуль управления коллекциями для проверки. Если проверка пройдена, то страница скачивается и сохраняется в корневую папку под именем N.html, где N - номер определяемый управлением коллекций. В противном случае запрашивается следующая ссылка.

Результатом работы алгоритма является папка HTML документов по теме запроса с номерами от 1 до заданного, без пропусков, что облегчает последующее к ним обращение.

## 2.3. Декодирование символов из UTF8 в CP1251 и обратно.

CP1251 – это однобайтная кодировка, что позволяет ей хранить значения 256 символов [11]. Первые 128 символов соответствуют стандарту ASCII (American standard code for information interchange) [12]. Бит чётности, который раньше использовался для проверки ошибки при передаче информации, в CP1251 расширяет таблицу ASCII ещё на 128 символов, среди которых русские заглавные и строчные буквы, а так же некоторые ранее отсутствовавшие знаки препинания. Данная кодировка была разработана для «самопальной» локализации Windows, но она прижилась и теперь достаточно широко распространена в русском сегменте интернета.

UTF8 (от англ. Unicode Transformation Format, 8-bit — «формат преобразования Юникода, 8-битный») — одна из общепринятых и стандартизированных кодировок текста, которая позволяет хранить символы Юникода, используя переменное количество байт (от 1 до 6). Основным преимуществом UTF8 является совместимость с ASCII — любые их 7-битные символы отображаются как есть, а остальные выдают пользователю

мусор (шум). Поэтому в случае, если латинские буквы и простейшие знаки препинания занимают существенный объём текста, UTF8 даёт выигрыш по объёму по сравнению с юникодом, все символы которого занимают два байта.

Алгоритм перекодирования символа UTF8 в CP1251 следующий.

1). Так как заранее не известно сколько байт используется для кодирования конкретного символа, то входным значением является положение символа в строке.

2). Если значение первого байта меньше 128, то он возвращается как есть.

3) Если значение выше 128, то определяется, сколькими байтами кодируется символ. По стандарту UTF8 первый байт содержит в старших разрядах столько единиц, сколькими байтами кодируется символ, с последующим нулём. К примеру для двухбайтного символа первый байт будет выглядеть так 110XXXXX, где X – значащие биты символа. Далее все символы объединяются в одно числовое значение путём сдвига байтов влево, так чтобы они не перекрывали друг друга.

4). Полученное значение сравнивается с таблицей длинной 128 ячеек. Если найдено совпадение, то символ CP1251 соответствующий исходному имеет значение N+128, где N – номер ячейки таблицы. В противном случае исходный символ отсутствует в CP1251 и возвращается значение символа, заданного в настройках для отсутствующих символов.

Алгоритм обратного преобразования практически такой же за исключением того, что из таблицы сразу берётся значение ячейки соответствующей номеру символа в CP1251, а полученная последовательность байт выдаётся в виде строки.

## 2.4. Выделение предложений.

В данном исследовании интерес представляют только частоты пар

слов, использованных в одном предложении, поэтому нужно выделить их из документа, попутно очистив текст от HTML кода. Так как ссылочное оглавление, реклама и списки редко заканчиваются на точку или другие знаки встречающиеся в конце предложения, то упомянутые части Web-страницы, как правило, не попадают в обработанную страницу.

Выделение предложений делится на три этапа.

Первые этап – это определение кодировки документа:

1). Попытаться найти запись о кодировке в теге <meta>. Для этого сначала ищется сам тег, а затем внутри него запись «charset=». Если после неё следует «windows-1251», то текст передаётся далее без изменений. При обнаружении записи «utf-8» весь текст переводится в CP1251.

2). В том случае, когда запись о кодировке отсутствует или её не удалось распознать, программа пытается найти в документе русские слова длинной более 3 символов использованные в поисковом запросе, по которому скачан документ. Если ничего не найдено, то проводится перекодировка из UTF8 в CP1251 и поиск повторяется. При отсутствии совпадений кодировка считается неопределённой, и документ исключается из коллекции.

Во втором этапе используется распределение тегов HTML на 4 группы:

1). Пропускаемые – при встрече они пропускаются, ни на что не влияя: <a>, <abbr>, <acronym>, <b>, <bdi>, <bdo>, <big>, <blink>, , <cite>, <code>, <del>, <dfn>, , <font>, <i>, <img>, <ins>, <kbd>, <mark>, <nobr>, <plaintext>, <q>, <s>, <small>, , <strike>, , <sub>, <sup>, <time>, <tt>, <u>, <var>, <wbr>, <xmp\>.

2). Открывающие – включают запись следующего за ними текста, обозначают абзац, то есть если предыдущее предложение не завершено, оно исключается из документа: <address>, <article>, <blockquote>, <dd>, <details>, , <dt>, <footer>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, <li>, <main>, <p>, <td>, <th>.

3). Закрывающие – выключают запись текста. Это те же теги что и открывающие, только они означают конец действия тега: </address>, </article>, </blockquote>, и т. д.

4). Прерывающие – обозначают абзац, но запись после них не прекращается: <br>.

Все неопределённые теги работают как закрывающие.

Производится проход по тексту с использованием алгоритма конечного автомата с четырьмя состояниями. Переменные, указанные в состояниях означают:

- Data_write – записывать текст или нет.

- Space – необходимо вывести пробел перед следующим записываемым символом.

- In_sentence – означает, что сканирование находится внутри предложения.

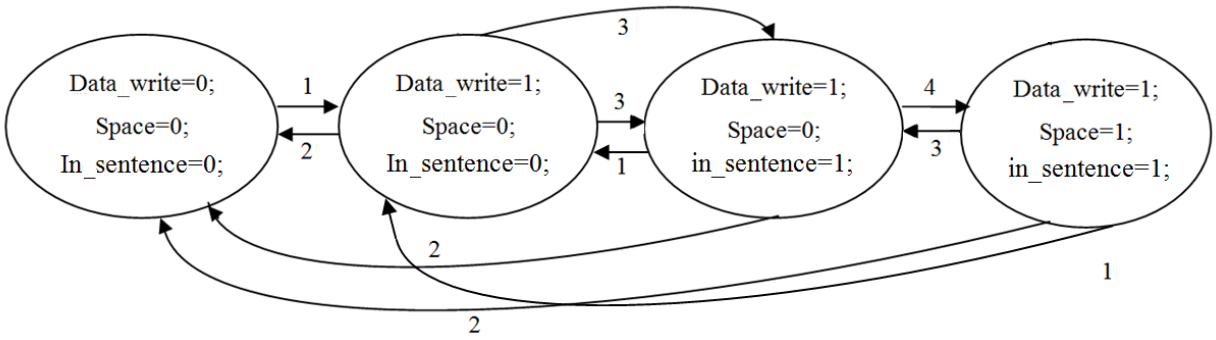


Рисунок 2.1. Конечный автомат для выделения предложений.

Стрелки с номерами означают переходы в другое состояние при возникновении конкретных представленных ситуаций:

1). Следующий тег на странице определён как открывающий или прерывающий. Это означает, что запись возможна. Сам тег записывается в результат для использования в третьем этапе.

2). Определён закрывающий тег. Текст не записывается.

3). Символ выводится в результат, а также это означает что началась запись предложения.

4). Попался пробельный символ, при переходе в состояние 3 выведется пробел.

Так как нет стрелки ведущей из состояния 2 в состояние 4, то предложение не будет начинаться с пробела. Если сканирование находится в состоянии 4 – пробел не записывается, следовательно все слова будут разделены не более чем одним пробелом.

На третьем этапе происходит второй проход по тексту с целью избавиться от того, что не является предложением.

1). Если встречается открывающий тег, записанный на предыдущем этапе, а предложение не было завершено, результат откатывается до предыдущего сохранённого предложения.

2). При встрече точки, восклицательного или вопросительного знака производится проверка на то, являются ли они концом предложения.

- Восклицательный и вопросительный знаки проходят проверку если за ними стоит пробел или тег.
- Точка обозначает конец предложения, если за ней стоит пробел, за которым следует заглавная буква или тире. Производится проверка, не обозначает ли точка сокращение инициала, путём проверки является ли предыдущий символ заглавным с пробелом перед ним.

Если полученное предложение начинается с маленькой буквы, цифры или символа кроме тире, то оно стирается, в противном случае оно сохраняется, а в конец добавляется символ переноса строки (\r\n).

3). Все остальные символы выводятся в результирующий текст, но это не значит, что они будут сохранены, это пояснено выше.

Таким образом, результатом обработки документа N.html является текстовой файл N.txt очищенный от HTML кода и, по-возможности, ссылочного оглавления и рекламы. К тому же, этот файл разбит на предложения. Каждое предложение начинается с новой строки.

Из дополнительных возможностей хочется отметить способность распознавать escape-последовательности (& >), соответствующие символам CP1251, написанные в любом возможном виде.

## 2.5 Лемматизация.

Для лемматизации используется готовое решение. Программе остаётся только разбить текст на токены. Это осуществляется за два прохода по тексту. В качестве символа разделителя используется пробел.

1). Обратный проход при котором от слов отделяются следующие знаки препинания: «!», «”», «)», «*», «,», «», «.», «:», «;», «?», «]», «|», «}», «,», «„», «…», «’», «”», «™», «›», «©», «®», «°», «»». Кажется что некоторые символы одинаковы, но на самом деле они имеют разные коды в CP1251. В результате выполнения прохода порядок символов в документе меняется на обратный.

2). Ещё один обратный проход, который вернёт порядок символов в исходный и добавит пробелы после символов: «"», «(», «[», «{», «|», «‘», «“», «‹», «‹‹», «–», «—».

В CP1251 есть одно особенное значение с кодом 152. Ему не присвоено никакого символа. Первоначально казалось, что можно отсутствующие в кодировке символы заменить на него, но данный символ не обрабатывается библиотекой лемматизатора. Поэтому при обработке он удаляется с проверкой, не остались ли после удаления лишние пробелы.

Далее текст поступает в функцию лемматизатора по предложениям. Результатом работы данного модуля программы является файл N_lem.txt, где N номер исходного файла. Весь текст содержащийся в нём лемматизирован, а каждое предложение как и раньше начинается с новой строки.

## 2.6. Сравнение строк с помощью хэш таблиц.

Сравнение двух строк - операция несложная. Но если необходимо

часто определять совпадает ли исходная строка с одной из сотен тысяч, это может вылиться в очень продолжительное время ожидания. Для эффективного решения данной задачи используются хэш таблицы.

Основная идея алгоритма в предварительном вычислении хэш функции от всех строк и хранения ссылок на них в ячейках хэш таблицы с номерами равными значению хэш функции делённому на длину таблицы. Таким образом, для исходной строки нужно вычислить хэш функцию и можно будет сравнивать строку не со всеми ста тысячами, а только с теми что находятся в соответствующей ячейке.

Так как точность исследования повышается с размером коллекции и охвате тем, был разработан алгоритм хранения данных, позволяющий записать экстремально большое количество строк и соответствующих им значений.

Строки, с которыми происходит сравнение хранятся на жёстком диске в папке данных хэша с именами N_F.txt, где N – длина строк помещаемых в данный файл, а F – номер файла. Такой способ хранения позволяет снизить объём данных в оперативной памяти, потому что ссылкой является не положение первого символа строки в файле, а её номер. Максимальное количество строк равно 65535, так как это самое большое число, кодируемое двумя байтами. Чтобы сохранить больше строк заданной длины нужно повысить номер файла. Список номеров файлов для всех длин во время выполнения программы хранится в оперативной памяти и после выполнения записывается на диск. Его объём достаточно мал, потому что в данном исследовании самой длинной строкой является предложение, а за весь сбор коллекции самая длинная строка равнялась 7632 символам, так как номер файла тоже записывается двумя байтами, общий размер получается чуть менее 16 килобайт.

Файлы для хранения строк имеют следующую структуру.

| **Строка** | **Табуляция** | **Ссылочное значение** | **Перенос строки** |
|---|---|---|---|
| str | \t | 4-х байтное число | \r\n |

Таблица 2.1. Структура файлов для хранения строк.

Ссылочное значение всегда присутствует, но задавать его не обязательно. Оно необходимо к примеру в тех случаях когда нужно определить номер слова в словаре коллекции. Так как после сравнения строк результатом является это число, оно должно быть больше 0. Ведь нулевое значение означает отсутствие строки в базе хэша.

Данные о строках записываются в ячейки хэш таблицы в виде трёх двухбайтных значений: длина строки, номер файла и номер строки в файле. Таблица имеет следующий вид:

| **длина** | **№ файла** | **№ строки** | **длина** | **№ файла** | **№ строки** | **длина** | **№ файла** | **№ строки** |
|---|---|---|---|---|---|---|---|---|
| 100 | 0 | 11 | | | | | | |
| | | | | | | | | |
| 8 | 0 | 52343 | 53 | 0 | 5193 | 8 | 1 | 1035 |
| 15 | 0 | 25123 | 636 | 0 | 1 | | | |
| | | | | | | | | |
| 5 | 0 | 768 | | | | | | |

Таблица 2.2. Пример хэш таблицы.

Хранение строк одинаковой длины удобно ещё и потому, что при сравнении строк попавших в определённую строку хэш таблицы можно сразу же исключить строки длины которых не совпадают.

Большинство ячеек таблицы из примера пусты, так как память выделяется только под хранение данных. Расширить заранее выделенное количество памяти невозможно из-за того, что данные в памяти хранятся последовательно. Гарантировать наличие необходимого количества

свободных ячеек в выделенной памяти невозможно. Поэтому в таких ситуациях необходимо пересоздавать массивы с большей длиной и копировать в них предыдущие данные. Так работает расширение строк хэш таблицы и массива номеров файлов, в которые помещается строка определённой длины. Знать заранее их размеры невозможно.

Строки хэш таблицы имеют переменную длину. Для её хранения используется массив двухбайтных переменных размером с длину хэш таблицы. Необходимо хранить указатели (занимающие 4 байта в 32 битных системах) на память, в которой хранятся её строки, таким образом память для хранения всей хэш таблицы равна:

$$M=6N+(2+4)L$$

где M – память, N – количество записей и L – длина хэш таблицы.

Самой оптимальной хэш таблицей будет та, в которой в каждой строке хранится не более одного значения. Тогда для определения присутствия исходной строки в базе данных понадобится всего одно сравнение. Один из основных факторов влияющих на эффективность – длина хэш таблицы. По данным [15] оптимальная длина хэш таблицы – это простое число в 1.5 раза большее предполагаемого количества строк для поиска. Неплохим выбором будет не простое число, а $2^n$-1.

Для хеширования используются две хэш функции в зависимости от типа данных строк:

1). Хэш функция для строк символов равна числу составленному из битов в позициях:

$$p = \frac{L * I}{32}$$

где p – позиция бита, L – длина строки в битах, I=1, … , 32.

2). Для числовых строк, где числа представляют собой номера слов в словаре, а значения старших трёх байт слабо изменяются от слова к слову, используется такая же функция. Но позиции бит, попадающих в вычисляемое

значение, могут находиться только в первом байте каждого числа из строки. Если таких байт не достаточно для составления значения, то используются остальные.

При завершении работы хэша, чтобы не пришлось заново пересчитывать все его значения, таблица выводится на жёсткий диск. Для ситуации, когда необходимо увеличить длину хэша, в программе реализован проход по базе данных хеш значений с необходимыми вычислениями.

## 2.7. Кодирование документов.

После лемматизации документа необходимо выполнить его кодирование.

Код слова соответствует его номеру в словаре, который составляет данный модуль. Для определения какие слова уже есть, а каких нет используется сравнение с помощью хэш таблиц. Для уменьшения объёма словаря, все слова содержащие символы не из кириллицы исключаются. Также ведётся подсчёт количества встреч слов, чтобы впоследствии уменьшить размер частотного словаря пар слов.

Таким образом, предложения записываются в кодированный файл в виде последовательности четырёхбайтных целых чисел без знака. Конец предложения обозначается нулём. Если в результате кодирования оказывается, что предложение содержит менее двух слов, то оно исключается, потому что при составлении частотного словаря такое предложение не даст никаких данных.

Перед записью номера слов в предложении сортируются, что позволяет проверить их на плагиат, без учёта перестановки слов. Плагиат не сохраняется, чтобы не завышать частоты пар слов при подсчёте частотного словаря.

Результат работы – файл N_code.txt, где N номер исходного файла. Пользователю будет сложно прочитать содержимое данного файла. Так как

он выводится в виде понятном компьютеру, во избежание лишних преобразований.

## 2.8. Управление коллекцией.

Модуль управления коллекциями служит для хранения статистических данных об обучающей коллекции, а также управляет этапами обработки документов и темами для запросов к поисковой системе.

| Название коллекции | Кол-во тем | Кол-во документов | Макс. Документов на тему | Исп. ссылок | Слов | предложений |
|---|---|---|---|---|---|---|
| "MyCollection" | 1134 | 56700 | 50 | 71916 | 4681693 | 643350 |

Таблица 2.3. Запись о коллекции

Записи о коллекциях хранятся в файле collections.txt.

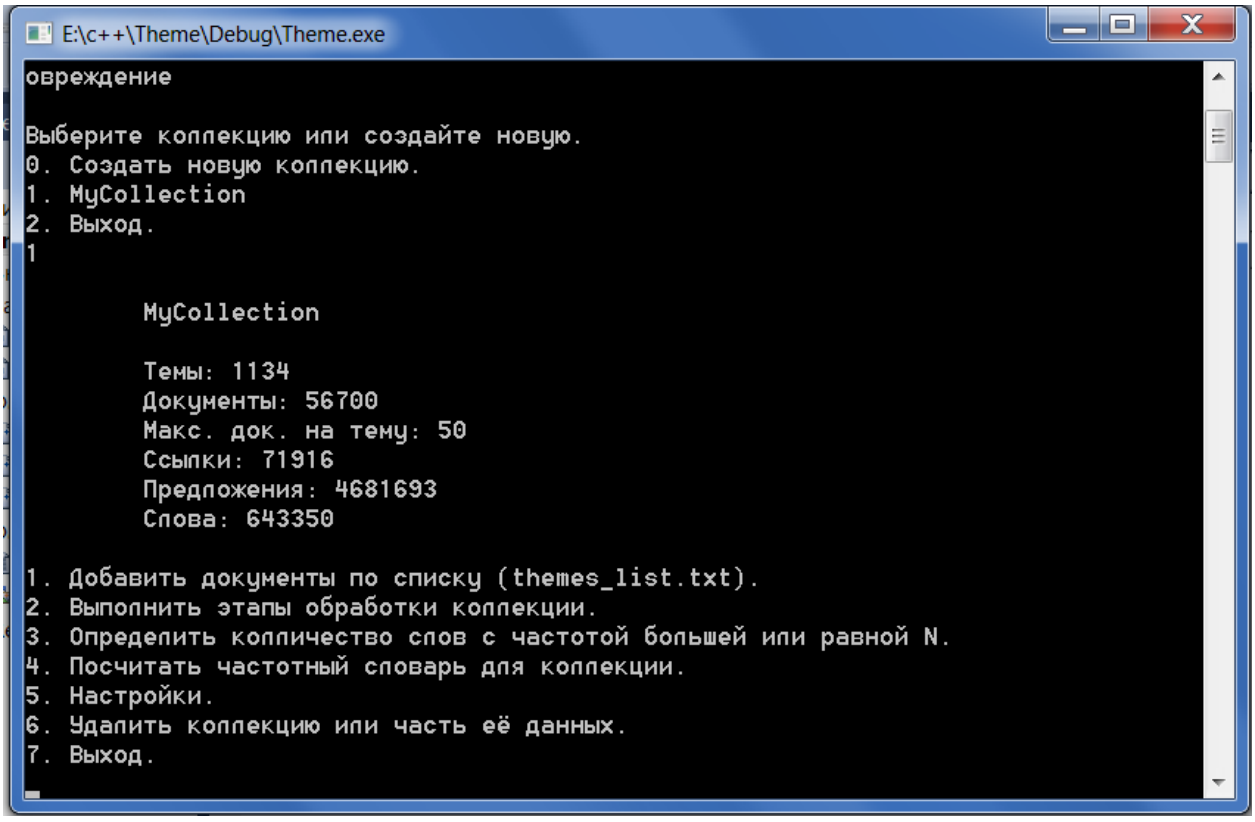


Рисунок 2.2. Меню программы для работы с коллекциями.

Основной функцией модуля коллекции является добавление и обработка документов по записанным пользователем темам запросов из файла theme_list.txt, находящегося в корневой папке коллекции. Алгоритм

работы данной функции следующий:

1). Предварительно выполняется обработка списка тем с удалением лишних пробелов и комментариев пользователя и определяется их количество.

2). Далее пользователь может задать: с какой по какую темы использовать и сколько документов скачивать по каждой из них.

3). С помощью хэш таблицы определяется номер темы или присваивается новый. Все скачанные и обработанные документы находятся в папке data. В папках с номерами, соответствующими номеру темы.

4). В папке с темой расположен файл properties, в котором указано количество скачанных по теме документов и сама тема. Если документов меньше, чем задал пользователь для сбора или тема новая - добавляется необходимое количество с последующей обработкой.

5). В процессе загрузки модуль проверяет ссылки, по которым скачиваются документы, чтобы исключить попадание в коллекцию одинаковых документов и не работать с серверами из списка исключений (server_except_list.txt).

В модуле сбора коллекции также присутствуют функции для удаления результатов любого из этапов и его повторного выполнения.

Чтобы уменьшить размер словаря частот пар слов необходимо выбрать частоту, начиная с которой, слова попадут в словарь. Для этого в модуле сбора коллекции есть функция, которая по заданному пользователем значению показывает сколько слов в словаре коллекции с ним встречалось.

В корневой папке коллекции находятся каталоги links_hash, word_hash, theme_hash и sentence_hash с данными соответствующих хэшей. Помимо этого в ней хранится директория frequency_vocabulary содержащая таблицу частотного словаря.

## 2.9. астройки.

Каждая настройка хранит своё имя, максимальное, минимальное, значение по умолчанию, а также значение, заданное пользователем. Все они помещаются в список, что позволяет сохранить, загрузить или вернуть к значению по умолчанию их все. Все это делается с помощью одной команды, что облегчает работу программиста.

В программе присутствуют следующие настройки:

1), Использовать Яндекс XML или его стандартный поиск.

2). Опции запроса для Яндекс XML (составляются в соответствии с документацией [10])

3). Использовать точный запрос – Яндекс будет искать точно то, что указано в теме запроса не пытаясь исправить орфографию.

4). Сохранять ссылки. Они будут сохранены сначала каждой скачанной HTML страницы между тегами <selflink>.

5). Использовать список исключений серверов.

6). Есть возможность отключить этапы обработки при добавлении документов в коллекцию, если они не нужны.

7). Минимальная длина предложения. При выделении предложений, если длина меньше заданного минимума символов, то предложение не сохраняется.

8). Минимальное количество слов в предложении. Работает по аналогии с предыдущим вариантом, но для слов.

9). Сохранять открывающие абзац теги для того, чтобы сохранить данные о том, какие предложения находились в одном абзаце. Эти данные впоследствии могут улучшить результаты эксперимента. Сейчас они не используются и полностью теряются после кодирования предложений.

10). Минимальное количество предложений в документе – документы с меньшим количеством предложений не попадают в коллекцию.

11). Сортировать слова в предложении – можно отключить сортировку, если важен порядок слов. Правда это нарушит работу определения плагиата.

12). Проверку предложений на плагиат – можно отключить. Используется для снижения потребления памяти и сокращения времени выполнения.

13). Задать символ которым заменяются отсутствующие в CP1251.

14). Есть возможность отключить работу любого хэша, кроме хэша слов. Соответственно функции, зависящие от него не будут выполняться.

15). Можно задать вручную длину любой хэш таблицы. Если заранее известно количество строк, которые попадут в хэш, то желательно установить эти значения вручную. Так как никакой проверки на эффективность установленных длин не производится, вся ответственность за выбор лежит на пользователе.

16). Максимальный объём памяти, для использования в хэш таблицах. Задаётся для того, чтобы программа не вызвала ошибку памяти.

17). Минимальная частота встречи слова для попадания в словарь частот пар слов.

18). Максимальный объём памяти для частотного словаря – чем больше данных частотный словарь хранит в оперативной памяти, тем быстрее он работает.

# Глава 3. лгоритмы подсчёта частот и поиска аналогов документа.

## 3.1. Подсчёт частотного словаря пар слов.

Данные, получаемые на этом этапе, имеют очень широкое применение и представляют основную ценность данного исследования. Поиск аналога документа – это всего лишь одна из прикладных задач, которую они помогают решить.

Для хранения частот встречи пар слов в одном предложении необходима таблица NxN, где N число слов, которые в неё попадают. Вот пример данной таблицы:

| | **Слово_1** | **Слово_2** | **Слово_3** | **Слово_4** | **Слово_5** |
|---|---|---|---|---|---|
| **Слово_1** | 5 | 1 | 4 | 0 | 2 |
| **Слово_2** | 1 | 10 | 7 | 0 | 3 |
| **Слово_3** | 4 | 7 | 65535 | 3 | 75 |
| **Слово_4** | 0 | 0 | 3 | 7 | 0 |
| **Слово_5** | 2 | 3 | 75 | 0 | 120 |

Таблица 3.1. Пример таблицы частот слов.

Очевидно, таблица будет симметрична относительно диагонали. Это могло бы позволить сократить объём памяти, необходимой для её хранения в два раза. Из-за того, что таблица хранится на жёстком диске, это потребовало бы для последних строк собирать значения из случайно расположенных блоков памяти. Такой подход неприемлем, вследствие, слишком большой задержки поиска данных на диске. Для редко встречавшихся в коллекции слов, сумма значений в их строках будет равна нулю. Казалось бы, можно сократить потребление памяти, путём разработки специального алгоритма хранения, не учитывающего нули. Но с ростом обучающей коллекции этот алгоритм был бы уже не столь эффективен,  а так как точность исследования

повышается с ростом коллекции, решено было отказаться от его разработки. Чтобы уменьшить потребление памяти, частоты записываются двухбайтным целым числом без знака. Точность, которую оно даёт, приемлема для эксперимента.

Словарь для частот пар слов уменьшается с помощью исключения слов, встречающихся при сборе коллекции с малой частотой. Но для таких слов впоследствии необходимо знать номер слова в словаре кодирования. Для этого составляются связанные списки. Один хранит указатели на другой, а тот в свою очередь указывает на первый. Значения для слов в длинном списке, отсутствующих в коротком, равны нулю.

Чтобы сократить количество обращений к диску, пользователь может указать объём оперативной памяти, которая будет использована для хранения строк таблицы соответствующих самым часто используемым словам в обучающей коллекции. Для этого сортируется список частот слов и определяется, с какой частотой слова попадут в оперативную память. Также создаются соответствующие связанные списки между словами в словаре частот и словами, попадающими в оперативную память.

Алгоритм подсчёта частот пар слов в одном предложении следующий:

1). Используется входной кодированный файл. Из него поступают предложения в виде чисел. Конец предложения обозначен нулём.

2). Слова, не попавшие в частотный словарь исключаются, остальные меняют своё значение на номер в частотном словаре. Благодаря тому, что предложения отсортированы, при кодировании из них убираются повторяющиеся слова, чтобы не завышать частоты пар их встреч.

3). В полученном предложении осуществляется перебор всевозможных пар слов с повышение соответствующих частот в таблице.

Таблица частот хранится в папке frequency_vocabulary в корневой папке коллекции. Файлы с именами N соответствуют строке таблицы.

## 3.2. Поиск документов аналогов.

Чтобы выполнить поиск документов аналогов пользователю необходимо поместить исходный текст в файл analog_in.txt в указать, данные из какой коллекции использовать. Так как не предусмотрено никакой проверки длины словаря, не допускается проводить добавление или кодирование документов обучающей коллекции после составления словаря частот иначе программа завершится с ошибкой. Если добавление было выполнено или было изменено значение частоты при котором слово попадает в частотный словарь, необходимо удалить данные таблицы частот, и составить таблицу повторно.

Исходный документ, по которому пользователь собирается найти аналоги, проходит три стандартные этапа обработки:

1). Выделение предложений – выполняется модулем, написанным для выделения предложений из HTML страниц. Из-за этого, чтобы не отрезать слова в конце документа, в конце должна стоять точка.

2). Лемматизация – работает без ограничений.

3) Кодирование – при кодировании неизвестные коллекции слова добавляются в словарь с частотой 0.

Данная обработка занимает больше времени, чем хотелось бы (30 секунд), так как почти для каждого этапа необходимо загрузить, сохранить и закрыть необходимые ему базы данных. Этого можно избежать, если искать аналоги для нескольких документов сразу, но пока данный метод не реализован.

Полученный кодированный документ сортируется целиком в порядке понижения номеров слов. С помощью этого, можно посчитать частоту слов исходного документа объединив одинаковые слова.

Результатом является список всех слов документа с указанием их частот. Далее для них вычисляются следующие функции, полученные

эмпирическим путём:

1). Известность – частота встречи слова при подсчёте частотного словаря делёная на 100. Установлено ограничение, не позволяющее значению данной функции превышать единицу.

2). Уникальность – для её подсчёта используются два счётчика. Осуществляется проход по строке таблицы частот соответствующей исходному слову, во всех положениях, известность слов которых больше или равна 25%. Если частота встречи данных слов больше нуля, то увеличиваются оба счётчика, противном случае только один. Значение функции равно частному этих счётчиков вычтенному из единицы. Если слова нет в частотном словаре, то функция равна единице.

3). Принадлежность – определяет, насколько слово принадлежит исходному тексту. Для неё так же используются два счётчика. Осуществляется проход по строке таблицы частот соответствующей слову, для которого вычисляется функция в позициях с частотами слов из документа с уникальностью более 50%. Если данные слова употреблялись в одном предложении, то увеличиваются оба счётчика. Если нет то только один. Значение функции равно частному счётчиков. В случае отсутствия слова в частотном словаре функция равна 0.

Все значения функций используются при рассчёте релевантности в качестве переменных, по следующей формуле:

$$R=FU^2(1+I+P)$$

Где R – релевантность, F – частота в исходном документе, U – уникальность, I – известность, P – принадлежность. Уникальность возводится в квадрат, чтобы понизить уникальность слов используемых при описании любой темы, но не сильно повлиять на уникальность остальных слов.

Далее список слов сортируется в порядке снижающейся релевантности и данные выводятся в файл. В нём пользователь сможет найти 40 слов с наибольшими значениями релевантности и данные, позволяющие оценить

работу программы. Формат вывода данных будет подробнее рассмотрен в главе результаты.

# Глава 3. Результаты.

В файле с результатами содержится весь список слов документа с вычисленными для него функциями. Так как он слишком велик чтобы разместить его в тексте работы, то будут написаны только первые десять слов с наиболее высокой релевантностью.

Для тестирования было выбрано по одному документу из каждой области тем для запросов. Чтобы проиллюстрировать работу программы на неохваченных областях был запущен поиск аналогов для текста данной работы.

| № | Слово | Частота | Известность | Уникальность | Принадлежность | Релевантность |
|---|---|---|---|---|---|---|
| 1 | математика | 61 | 1 | 0,863491 | 0,919156 | 132,771 |
| 2 | математический | 30 | 1 | 0,848056 | 0,931459 | 63,2491 |
| 3 | теория | 15 | 1 | 0,633322 | 0,949033 | 17,7427 |
| 4 | обьект | 16 | 1 | 0,611997 | 0,947276 | 17,662 |
| 5 | наука | 15 | 1 | 0,632411 | 0,954306 | 17,2225 |
| 6 | править | 15 | 1 | 0,578558 | 0,919156 | 14,6569 |
| 7 | геометрия | 6 | 1 | 0,907409 | 0,838313 | 14,0222 |
| 8 | вики-текст | 14 | 0 | 1 | 0 | 14 |
| 9 | изучение | 9 | 1 | 0,698398 | 0,929701 | 12,8609 |
| 10 | элементарный | 6 | 1 | 0,861071 | 0,845343 | 12 ,658 |

Таблица 4.1. Результаты для документа из области «наука».

Первый документ из области «наука». Он получен по запросу «математика» и содержит статью из Википедии о математике. Как видно из таблицы, программа смогла подобрать достаточно релевантные слова для запроса. Присутствие слов в разных формах «математика» «математический» можно рассматривать как достоинство или недостаток в зависимости от алгоритма лемматизации поисковой системы, которой будет отправлен данный запрос. Если она не распознает их как одно слово, то релевантность запроса повысится. В противном случае релевантность запроса будет такой же как с одним словом «математика». Слова «править» и «вики-текст» попали в таблицу из-за того, что текст был скопирован с HTML без

редактирования. Эти слова не принадлежат тексту статьи.

| о№ | Слово | Частота | Известность | Уникальность | Принадлежность | Релевантность |
|---|---|---|---|---|---|---|
| 1 | самуэльсон | 62 | 1 | 0,981762 | 0,497627 | 149,256 |
| 2 | теория | 51 | 1 | 0,633322 | 0,962816 | 60,6072 |
| 3 | экономический | 68 | 1 | 0,527098 | 0,968354 | 56,08 |
| 4 | самуэльсонный | 27 | 0,72 | 0,989338 | 0,252373 | 52,1245 |
| 5 | анализ | 34 | 1 | 0,644101 | 0,935918 | 41,4125 |
| 6 | предпочтение | 18 | 1 | 0,862173 | 0,775316 | 37,1341 |
| 7 | доход | 26 | 1 | 0,69169 | 0,912184 | 36,2256 |
| 8 | предельный | 17 | 1 | 0,857233 | 0,794304 | 34,9077 |
| 9 | модель | 25 | 1 | 0,690489 | 0,919304 | 34,7963 |
| 10 | благосостояние | 16 | 1 | 0,885934 | 0,726266 | 34,2366 |

Таблица 4.2. Результаты для документа из области «экономика».

Второй документ из области «экономика». Получен по запросу «американская экономическая ассоциация». Содержит реферат «Теории Пола Семюельса». Слово «самуэльсонный» лемматизировано неправильно. Это слово явно не словарное, а так как автор библиотеки лемматизатора упоминал о точности в 99%, то оно находится в области допустимой погрешности.

| № | Слово | Частота | Известность | Уникальность | Принадлежность | Релевантность |
|---|---|---|---|---|---|---|
| 1 | маврикий | 34 | 1 | 0,93464 | 0,737226 | 81,2978 |
| 2 | император | 27 | 1 | 0,566461 | 0,966423 | 25,7002 |
| 3 | полководец | 12 | 1 | 0,804872 | 0,846715 | 22,1299 |
| 4 | сочинение | 11 | 1 | 0,805874 | 0,823358 | 20,1694 |
| 5 | искусство | 13 | 1 | 0,702319 | 0,880292 | 18,4692 |
| 6 | перевод | 10 | 1 | 0,759853 | 0,814599 | 16,2508 |
| 7 | славянин | 7 | 1 | 0,857534 | 0,814599 | 16,2508 |
| 8 | византийский | 8 | 1 | 0,777257 | 0,893431 | 13,984 |
| 9 | писатель | 7 | 1 | 0,97411 | 0,794161 | 12,3342 |
| 10 | военный | 24 | 1 | 0,406007 | 0,954745 | 11,6896 |

Таблица 4.3. Результаты документа из области «история».

Третий документ из области «история». Получен по запросу «свержение императора маврикия в византии». Содержит предисловие книги

Маврикия «Тактика и стратегия». Все слова в таблице достаточно релевантны.

| № | Слово | Частота | Известность | Уникальность | Принадлежность | Релевантность |
|---|---|---|---|---|---|---|
| 1 | чешский | 22 | 1 | 0,873152 | 0,727794 | 45,7524 |
| 2 | ес | 11 | 1 | 0,815902 | 0,882521 | 21,1077 |
| 3 | чехия | 10 | 1 | 0,871784 | 0,740688 | 20,8294 |
| 4 | нато | 10 | 1 | 0,803788 | 0,876791 | 18,5862 |
| 5 | запад | 16 | 1 | 0,628166 | 0,936963 | 18,5425 |
| 6 | россия | 39 | 1 | 0,378959 | 0,985673 | 16,7221 |
| 7 | экономический | 18 | 1 | 0,527098 | 0,949857 | 14,7522 |
| 8 | китай | 9 | 1 | 0,7145 | 0,905444 | 13,3493 |
| 9 | сша | 14 | 1 | 0,551911 | 0,959885 | 12,6224 |
| 10 | ли | 19 | 1 | 0,473002 | 0,952722 | 12,5517 |

Таблица 4.4. Результаты документа из области «политика».

Четвёртый документ из области «Политика». Получен по запросу «дестабилизация ситуации в россии невыгодна западу». Содержит интервью с Иво Стрейчек на тему «Запад стремится создать хаос в России». Слова кажутся не особенно релевантными, что подтверждает изначальную идею, о том, что научные статьи и определения словарей будут давать лучшие результаты чем публицистика или беллетристика. В таблицу попало слово «ли», но его там быть не должно, потому что оно должно иметь малое значение уникальности. Это происходит потому, что размер обучающей коллекции слишком мал и слово встречалось несколько раз, чтобы составить о нём правильное представление.

| № | Слово | Частота | Известность | Уникальность | Принадлежность | Релевантность |
|---|---|---|---|---|---|---|
| 1 | Запрос | 41 | 1 | 0.846638 | 0.805333 | 82.4449 |
| 2 | Документ | 63 | 1 | 0.626247 | 0.901333 | 71.6853 |
| 3 | Алгоритм | 25 | 1 | 0.889304 | 0.8 | 55.3603 |
| 4 | Коллекция | 25 | 1 | 0.913866 | 0.528 | 52.7816 |
| 5 | Частота | 25 | 1 | 0.862039 | 0.758667 | 51.2499 |
| 6 | Суффикс | 21 | 1 | 0.978058 | 0.206667 | 44.3287 |
| 7 | Таблица | 25 | 1 | 0.8284 | 0.848 | 42.9974 |
| 8 | Символ | 65 | 1 | 0.772351 | 0.844 | 42.413 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 9 | Поиск | 20 | 1 | 0.406574 | 0.944 | 31.6323 |
| 10 | Предложение | 22 | 1 | 0.707325 | 0.869333 | 28.7111 |

Таблица 4.5. Результаты по тексту данной работы.

Слова для последнего документа подобраны достаточно неплохо, но шанс нахождения документов с такой темой маловероятен поэтому они будут считаться релевантными если найдутся документы про похожие алгоритмы.

Далее приведена таблица релевантности найденных документов. Документы считаются релевантными, если их содержание схоже с содержанием исходного документа. В запросе использовались первые 5 и первые 10 слов. Релевантность определялась как частное количества релевантных документов к проверенным.

| | 5 слов в запросе | | | 10 слов в запросе | | |
|---|---|---|---|---|---|---|
| № док. | 10 | 20 | 30 | 10 | 20 | 30 |
| 1 | 0,7 | 0,7 | 0,66 | 0,7 | 0,75 | 0,73 |
| 2 | 0,8 | 0,75 | 073 | 1 | 0,95 | 0,9 |
| 3 | 0,9 | 0,75 | 0,76 | 0,7 | 0,65 | 0,63 |
| 4 | 0,9 | 0,85 | 0,8 | 0,7 | 0,5 | 0,46 |
| 5 | 0,9 | 0,85 | 0,8 | 0 | 0,2 | 0,33 |

Таблица 4.6. Релевантность найденных документов.

# Глава 4. Выводы.

Значение релевантности полученных документов частично доказывает гипотезу о том, что идея должна быть описана определёнными словами. Конечно, возможно, что релевантность немного завышена, так как сложно представить какую информацию хочет найти пользователь по исходному документу, но всё-таки она достаточно высока. Как минимум по каждой теме нашёлся документ, служивший исходным.

Все модули разработанного программного комплекса хорошо справляются со своими задачами. В дальнейшем необходимо улучшить работу Выделения предложений добавив эвристические функции для обнаружения рекламы и ссылок, а так же улучшить поиск аналогов документа точнее подобрав его функции. Также необходимо собрать коллекцию большего размера, с охватом большинства тем, что улучшит точность данных.

Программный комплекс слабо применим на пользовательских компьютерах ввиду его очень большого размера. Скорее он должен размещаться на серверах и выдавать пользователям информацию удалённо без необходимости хранения на жёстком диске.

# Заключение.

В ходе написания данной работы было рассмотрено применение данных о частоте встрече пар слов в предложении для поиска аналогов документа.

Были разработаны следующие модули программного комплекса:

1). Сбор коллекции путём скачивания документов по заданным темам.

2), Выделение предложений и очистка от HTML кода.

3). Лемматизация.

4). Кодирование документов и составление нумерованного словаря.

5). Составление частотного словаря встречи пар слов в предложении.

6) Применение частотного словаря для поиска аналогов документа.

Помимо этого были разработаны алгоритмы хранения данных хэш таблиц и собственная реализация алгоритма Бойера-Мура.

В дальнейшем планируется улучшить результаты исследования, а так же применить полученные данные частотного словаря для решения прикладных задач информационного поиска.

# Список литературы


1. Аликов А. Ю, Калиниченко А. В. Применение нечёткой кластеризации для автоматизации поиска похожих документов // «Вопросы науки и техники»: материалы международной заочной научно-практической конференции. Часть I. (16 января 2012 г.) — Новосибирск: Изд. «ЭКОР-книга» , 2012. С. 80-85.
2. Википедия, латентно семантический анализ. https://ru.wikipedia.org /wiki/%D0%9B%D0%B0%D1%82%D0%B5%D0%BD%D1%82%D0%BD%D0%BE-%D1%81%D0%B5%D0%BC%D0%B0%D0%BD%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7
3. Netpeak blog, Алгоритм LSA для поиска похожих документов. http://blog.netpeak.ru/algoritm-lsa-dlya-poiska-pohozhih-dokumentov/
4. Википедия, Количество статей в русской Википедии. https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D1%8F:%D0%9A%D0%BE%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D1%82%D0%B2%D0%BE_%D1%81%D1%82%D0%B0%D1%82%D0%B5%D0%B9_%D0%B2_%D1%80%D1%83%D1%81%D1%81%D0%BA%D0%BE%D0%B9_%D0%92%D0%B8%D0%BA%D0%B8%D0%BF%D0%B5%D0%B4%D0%B8%D0%B8
5. Компьютерная грамматика русского языка: лексика, морфология, синтаксис, Лемматизатор в грамматическом словаре. http://www.solarix.ru/for_developers/api/lemmatizator-api.shtml
6. Частотный словарь, Вторая версия частотного списка. http://www.artint.ru/projects/frqlist.php
7. CIS projects, TreeTagger - a part-of-speech tagger for many languages. http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger/

8. Википедия, Алгоритм Бойера — Мура. https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%91%D0%BE%D0%B9%D0%B5%D1%80%D0%B0_%E2%80%94_%D0%9C%D1%83%D1%80%D0%B0
9. Microsoft центр разработки для Windows, About WinINet. https://msdn.microsoft.com/ru-ru/library/windows/desktop/aa383630.aspx
10. Яндекс.XML, Документация, GET-запросы. https://tech.yandex.ru/xml/doc/dg/concepts/get-request-docpage/
11. Википедия, Windows-1251. https://ru.wikipedia.org/wiki/Windows-1251
12. Википедия, ACSII. https://ru.wikipedia.org/wiki/ASCII
13. Википедия, UTF-8 https://ru.wikipedia.org/wiki/UTF-8
14. Компьютерная грамматика русского языка: лексика, морфология, синтаксис, Грамматический Словарь Русского Языка. http://www.solarix.ru/index.shtml
15. Algolist.manual.ru, Хэш таблицы. http://algolist.manual.ru/ds/s_has.php

# Листинг

```
#include "stdafx.h"
#include <windows.h>
#include <wininet.h>
#pragma comment(lib,"wininet")
#pragma comment(lib,"RussianGrammaticalDictionary\\lib\\lemmatizator")
#include <iostream>
#include <fstream>
#include <locale>
#include
"RussianGrammaticalDictionary\include\lem\solarix\lemmatizator_engine.
h"

#pragma warning(disable: 4996)

using namespace std;

int compare(const void * x1, const void * x2)
{
     return ( *(int*)x1 - *(int*)x2 );
}
int reverse_compare(const void * x1, const void * x2)
{
     return ( *(int*)x2 - *(int*)x1 );
}
int short_compare(const void * x1, const void * x2)
{
     int a1=*(unsigned short int*)x1,
          a2=*(unsigned short int*)x2;
     return a2-a1;
}
int word_val_compare(const void * x1, const void * x2)
{
     double *a1=*(double **)x1,
          *a2=*(double **)x2;
     if(a2[5]==a1[5]) return 0;
     if(a2[5]>a1[5]) return 1;
     if(a2[5]<a1[5]) return -1;
}
char *revers(char *mass)
{
     unsigned int i,
          j,
          len=strlen(mass);
     if(len<2) return mass;
     char buf;
     for(i=0, j=len-1; i<j; i++, j--)
```

```
	{
		buf=mass[i];
		mass[i]=mass[j];
		mass[j]=buf;
	}
	return mass;
}
char *toa(int x, char *buf)
{
	unsigned int i=0,
		u_x=x<0?(-x):x;
	do{
		buf[i++]=u_x%10+'0';
	}while((u_x/=10)>0);
	if(x<0) buf[i++]='-';
	buf[i]='\0';
	return revers(buf);
}
char *toa(unsigned int x, char *buf)
{
	unsigned int i=0;
	do{
		buf[i++]=x%10+'0';
	}while((x/=10)>0);
	buf[i]='\0';
	return revers(buf);
}
char *toa(double x, char *buf)
{
	_gcvt(x, 10, buf);
	unsigned int point=strlen(buf)-2;
	if(buf[point]==',') buf[point]='\0';
	return buf;
}
int ato(char *buf, int &v)
{
	bool sign=*buf=='-'?1:0;
	unsigned int len=strlen(buf),
		i=0,
		u_v=0,
		j;
	if(len==0) throw buf;
	if(buf[i]=='+'||buf[i]=='-') i++;
	if(i==len) throw buf;
	while(buf[i]=='0'&&i<len) i++;
	if(i==len) return v=0;
	buf+=i;
	len-=i;
	if(len>10) throw buf;
```

```
	for(i=len-1, j=1; i>0; j*=10, i--)
	{
		if(buf[i]<'0'||buf[i]>'9') throw buf;
		u_v+=(buf[i]-'0')*j;
	}
	if((*buf<'0'||*buf>'9')||(len==10&&(*buf>'2'||(*buf=='2'&&(u_v>14
7483648||(u_v==147483648&&!sign))))))) throw buf;
	else u_v+=(*buf-'0')*j;
	return v=sign?-(int)u_v:u_v;
}
unsigned int ato(char *buf, unsigned int &v)
{
	unsigned int len=strlen(buf),
		i=0,
		j;
	v=0;
	if(len==0) throw buf;
	if(buf[i]=='+') i++;
	if(i==len) throw buf;
	while(buf[i]=='0'&&i<len) i++;
	if(i==len) return v;
	buf+=i;
	len-=i;
	if(len>10) throw buf;
	for(i=len-1, j=1; i>0; j*=10, i--)
	{
		if(buf[i]<'0'||buf[i]>'9') throw buf;
		v+=(buf[i]-'0')*j;
	}
	if((*buf<'0'||*buf>'9')||(len==10&&(*buf>'4'||(*buf=='4'&&v>29496
7295)))) throw buf;
	else v+=(*buf-'0')*j;
	return v;
}
double ato(char *buf, double &v)
{
	try
	{
		unsigned int len=strlen(buf),
			i=0,
			j=0;
		char *b=new char[len+1];
		if(len==0) throw b;
		if(buf[i]=='+'||buf[i]=='-') b[j++]=buf[i++];
		if(i==len) throw b;

	while(i<len&&buf[i]!=','&&buf[i]!='.'&&buf[i]!='e'&&buf[i]!='E')
		{
			if(buf[i]<'0'||buf[i]>'9') throw b;
```

```
                b[j++]=buf[i++];
            }
            if(i<len) if(buf[i]==','||buf[i]=='.')
            {
                b[j++]=(i++, ',');
                if(i==len) throw b;
                while(i<len&&buf[i]!='e'&&buf[i]!='E')
                {
                    if(buf[i]<'0'||buf[i]>'9') throw b;
                    b[j++]=buf[i++];
                }
            }
            if(i<len) if(buf[i]=='e'||buf[i]=='E')
            {
                b[j++]=buf[i++];
                if(i==len) throw b;
                if(buf[i]=='+'||buf[i]=='-') b[j++]=buf[i++];
                if(i==len) throw b;
                while(i<len)
                {
                    if(buf[i]<'0'||buf[i]>'9') throw b;
                    b[j++]=buf[i++];
                }
            }
            v=(b[j]='\0', strtod(b, NULL));
            if(errno) throw b;
            delete [] b;
            return v;
        }
        catch(char *buffer)
        {
            delete [] buffer;
            throw buf;
        }
}
void inmass(wchar_t *p, char *&mass, unsigned int &len, unsigned int
ext=0)
{
        ifstream in(p, ios::binary);
        if(!in) throw p;
        in.seekg(0, ios::end);
        len=(unsigned int)in.tellg();
        mass=new char[len+ext];
        in.seekg(0, ios::beg);
        in.read(mass, len);
        in.close();
}
void outmass(wchar_t *p, char *mass, unsigned int len)
{
```

```
	ofstream out(p, ios::out | ios::binary);
	if(!out) throw p;
	out.write(mass, len);
	out.close();
}
inline void crtdir(wchar_t *p)
{
	if(_waccess(p, 0)) if(!CreateDirectory(p, NULL)) throw p;
}
template<class type1> void endlessmass(type1 *&mass, unsigned int len,
unsigned int ext, unsigned int &maxlen)
{
	type1 *mass1;
	maxlen+=ext;
	mass1=new type1[maxlen];
	for(unsigned int i=0; i<len; i++) mass1[i]=mass[i];
	delete [] mass;
	mass=mass1;
}

//Настройки
class setting_base{
protected:
	char *name;
public:
	setting_base()
	{
		name=NULL;
	}
	char *get_name() {return name;}
	virtual void set(char *v)=0;
	virtual void def()=0;
	virtual char *note(char *buf)=0;
	~setting_base()
	{
		if(name!=NULL) delete [] name;
	}
};

template <class type> class setting:public setting_base{
	type value,
		def_value,
		min_value,
		max_value;
public:
	setting():setting_base() {}
	setting(char *n, type min, type max, type def)
	{
		initialize(n, min, max, def);
```

```
	}
	void initialize(char *n, type min, type max, type def)
	{
		name=new char[strlen(n)+1];
		strcpy(name, n);
		min_value=min;
		max_value=max;
		value=def_value=def;
	}
	void set(char *v)
	{
		type buf;
		ato(v, buf);
		if(buf<min_value||buf>max_value) throw v;
		value=buf;
	}
	void def() {value=def_value;}
	type get() {return value;}
	char *note(char *buf);
};
template <class type> char *setting<typename type>::note(char *buf)
{
	char b[30];
	strcpy(buf, name);
	strcat(buf, "=");
	strcat(buf, toa(value, b));
	strcat(buf, "\r\n");
	return buf;
}

template <> class setting<char *>:public setting_base{
	char *value,
		*def_value,
		*except;
	unsigned int max_len;
public:
	setting():setting_base()
	{
		value=NULL;
		def_value=NULL;
		except=NULL;
	}
	setting(char *n, char *e, unsigned int ml, char *def)
	{
		initialize(n, e, ml, def);
	}
	void initialize(char *n, char *e, unsigned int ml, char *def);
	void set(char *v);
	void def()
```

```
	{
		delete [] value;
		value=new char[strlen(def_value)+1];
		strcpy(value, def_value);
	}
	char *get(){return value;}
	char *note(char *buf);
	~setting()
	{
		if(value!=NULL) delete [] value;
		if(except!=NULL) delete [] except;
		if(def_value!=NULL) delete [] def_value;
	}
};
void setting<char *>::initialize(char *n, char *e, unsigned int ml,
char *def)
{
	unsigned int dl=strlen(def)+1;
	name=new char[strlen(n)+1];
	strcpy(name, n);
	except=new char[strlen(e)+1];
	strcpy(except, e);
	max_len=ml;
	def_value=new char[dl];
	strcpy(def_value, def);
	value=new char[dl];
	strcpy(value, def);
}
void setting<char *>::set(char *v)
{
	unsigned int len=strlen(v),
		elen=strlen(except),
		i,
		j;
	char *buf;
	if(len>max_len) throw v;
	buf=new char[len+1];
	for(i=0, j=0; i<len; i++, j++)
	{
		if(v[i]=='\\')
		{
			i++;
			if(v[i]=='r') buf[j]='\r';
			else if(v[i]=='n') buf[j]='\n';
			else if(v[i]=='\\') buf[j]='\\';
			else
			{
				delete [] buf;
				throw v;
```

```
                }
            }
            else buf[j]=v[i];
        }
        len=j;
        buf[len]='\0';
        for(i=0; i<len; i++) for(j=0; j<elen; j++) if(buf[i]==except[j])
        {
            delete [] buf;
            throw v;
        }
        delete [] value;
        value=new char[len+1];
        strcpy(value, buf);
        delete [] buf;
}
char *setting<char *>::note(char *buf)
{
        unsigned int len=strlen(value),
            i,
            j;
        strcpy(buf, name);
        strcat(buf, "=");
        for(i=0, j=strlen(buf); i<=len; i++, j++)
        {
            if(value[i]=='\r')
            {
                buf[j++]='\\';
                buf[j]='r';
            }
            else if(value[i]=='\n')
            {
                buf[j++]='\\';
                buf[j]='n';
            }
            else if(value[i]=='\\')
            {
                buf[j++]='\\';
                buf[j]='\\';
            }
            else buf[j]=value[i];
        }
        buf[j]='\0';
        strcat(buf, "\r\n");
        return buf;
}

class settings{
        unsigned int list_len,
```

```
            note_max_len;
      char *error;
      wchar_t *path;
      setting_base  **list;

      void add_error(char *n, unsigned int fl)
      {
            if(error==NULL)
            {
                  error=new char[note_max_len*list_len+fl+3];
                  *error='\0';
            }
            strcat(error, n);
            strcat(error, "\r\n");
      }
public:
      settings()
      {
            path=NULL;
            list=NULL;
            error=NULL;
      }
      settings(wchar_t *p, unsigned int c, unsigned int nml=258)
      {
            initialize(p, c, nml);
      }
      void initialize(wchar_t *p, unsigned int c, unsigned int
nml=258);
      void add(setting_base &s) {list[list_len++]=&s;}
      void def() { for(unsigned int i=0; i<list_len; i++) list[i]-
>def();}
      void save();
      char *load();
      void delete_error()
      {
            if(error!=NULL) delete [] error;
            error=NULL;
      }
      ~settings()
      {
            if(path!=NULL) delete [] path;
            if(list!=NULL) delete [] list;
            if(error!=NULL) delete [] error;
      }
};
void settings::initialize(wchar_t *p, unsigned int c, unsigned int
nml)
{
      unsigned int len=wcslen(p),
```

```
                i;
        path=new wchar_t[len+1];
        for(i=0; i<=len; i++)
        {
                if(p[i]=='\\')
                {
                        path[i]='\0';
                        crtdir(path);
                }
                path[i]=p[i];
        }
        list=new setting_base *[c];
        list_len=0;
        note_max_len=nml;
        error=NULL;
}
void settings::save()
{
        char *buf,
                *fl;
        buf=new char[note_max_len+1];
        fl=new char[note_max_len*list_len+1];
        strcpy(fl, list[0]->note(buf));
        for(unsigned int i=1; i<list_len; i++) strcat(fl, list[i]-
>note(buf));
        outmass(path, fl, strlen(fl));
        delete [] fl;
        delete [] buf;
}
char *settings::load()
{
        bool *d;
        unsigned int i,
                j,
                k,
                file_len,
                nml=note_max_len-2;
        char *note,
                *name,
                *val,
                *file;
        d=new bool[list_len];
        for(i=0; i<list_len; i++) d[i]=0;
        note=new char[note_max_len];
        inmass(path, file, file_len, 1);
        for(i=0; i<file_len; i++)
        {
                val=NULL;
```

```
            for(j=0; file[i]!='='&&file[i]!='\r'&&i<file_len&&j<nml;
i++, j++) note[j]=file[i];
            if(file[i]=='='&&i<file_len&&j<nml)
            {
                  i++;
                  note[j++]='=';
                  val=note+j;
            }
            for(; file[i]!='\r'&&i<file_len&&j<nml; i++, j++)
note[j]=file[i];
            if(file[i]!='\r'&&i<file_len)
            {
                  while(file[i]!='\n'&&i<file_len) i++;
                  note[nml-3]='\0';
                  strcat(note, "...");
                  add_error(note, file_len);
            }
            else
            {
                  note[j]='\0';
                  i++;
                  if(val==NULL) add_error(note, file_len);
                  else
                  {
                        for(k=0; k<list_len; k++)
                        {
                              if(d[k]) continue;
                              name=list[k]->get_name();
                              for(j=0; name[j]==note[j]; j++);
                              if(name[j]!='\0'||note[j]!='=') continue;
                              try
                              {
                                    list[k]->set(val);
                                    d[k]=1;
                              }
                              catch(char *v) {}
                              break;
                        }
                        if(!d[k]) add_error(note, file_len);
                  }
            }
      }
      for(k=0; k<list_len; k++) if(!d[k])
      {
            strcpy(note, list[k]->get_name());
            strcat(note, "=def");
            add_error(note, file_len);
      }
      delete [] file;
```

```
	delete [] note;
	delete [] d;
	return error;
}

// Поиск (по алгоритму Бойера Мура)
class finding_base{
protected:
	int alphabet[256], // Алфавит стоп символов.
		*suffix_table, // Таблица суффиксов, динамический массив.
		substrlen, // Длинна подстроки, которую ищем.
		pos, // Позиция поиска в документе, в котором ищем
подстроку.
		sourcelen; // Длинна документа, в котором ищем подстроку.
	char *substr, // Подстрока, динамический массив.
		*source; // Ссылка на документ, в котором ищем.

	void build_alphabet(); // Заполнение алфавита стоп символов.
	void build_suffix_table(); // Заполнение таблицы суффиксов.

public:
	finding_base()
	{
		substr=NULL;
		suffix_table=NULL;
	}

	void new_source(char *src, int sl) // Задаёт новый документ для
поиска в нём той же самой подстроки.
	{
		sourcelen=sl;
		source=src;
		pos=substrlen-1;
	}
	void set_position(int pn) {pos=pn+substrlen-1;} // Задаёт позицию
поиска.
	virtual int find()=0; //Производит поиск. При повторном вызове
продолжает поиск. Возвращает позицию начала слова или -1 если ничего
не нашла.

	~finding_base()
	{
		if(substr!=NULL) delete [] substr;
		if(suffix_table!=NULL) delete [] suffix_table;
	}
};
void finding_base::build_alphabet()
{
```

```
	for(int i=0; i<256; i++) alphabet[i]=substrlen; // Алгоритм
использует стоп символы перенося позицию следующего сравнения на
длинну слова, или до совмещения слова, со сравниваемым символом если
он входит в слово.
	for(int i=0; i<(substrlen-1); i++) alphabet[(unsigned
char)substr[i]]=substrlen-i-1; // i<(strlen(subStr)-1) потому что
последний символ подстроки не учитывается
}
void finding_base::build_suffix_table()
{
	//Таблица суффиксов используется, когда при сравнении хотя бы
один символ совпал, но один из последующих нет.
	int max_suffix=0,
		i,
		j;
	for(i=substrlen-1; i>0; i--)
	{
		for(j=1; j<=i; j++)
		{
			if(substr[substrlen-j]==substr[i-j]) // Ищем
совпадающий суффикс.
			{
				if(j>max_suffix)
suffix_table[++max_suffix]=substrlen-i; // Если совпавший на данный
момент суффикс больше известного максимального, то добавляем его в
таблицу.
				if(i==j)
				{
					i=substrlen-i;
					for(j=max_suffix+1; j<substrlen; j++)
suffix_table[j]=i;
					goto loop;
				}
			}
			else break;
		}
	}
	for(j=max_suffix+1; j<substrlen; j++) suffix_table[j]=substrlen;
// Для отсутствующих суффиксов происходит перенос на длинну слова.
	loop:;
}

class finding : public finding_base {
public:
	finding():finding_base() {}
	finding(char *str, char *src, int sl) // Принимает подстроку str,
которую будем искать в документе src с длинной sl.
	{
		initialize(str, src, sl);
```

```
	}
	void initialize(char *str, char *src, int sl);
	int find();
};
void finding::initialize(char *str, char *src, int sl)
{
	sourcelen=sl;
	source=src; //Сам документ должен где-то хранится пока мы
производим в нём поиск.
	substrlen=strlen(str);
	substr=new char[substrlen+1]; // на один символ больше для
хранения символа конца строки.
	suffix_table=new int[substrlen];
	strcpy(substr, str);
	pos=substrlen-1; //По алгоритму сравнение начинается с конца
подстроки.
	build_alphabet();
	build_suffix_table();
}
int finding::find()
{
	int i,
		j,
		sbl=substrlen-1;
	for(i=pos; i<sourcelen;)
	{
		if(source[i]==substr[sbl])
		{
			for(j=1; j<substrlen; j++) if(source[i-j]!=substr[sbl-
j])
			{
					i+=suffix_table[j];
					break;
			}
			if(j==substrlen)
			{
				pos=i+1;
				return i-sbl;
			}
		}
		else i+=alphabet[(unsigned char)source[i]];
	}
	pos=sourcelen;
	return -1;
}

class low_finding : public finding_base {
public:
	low_finding():finding_base() {}
```

```
      low_finding(char *str, char *src, int sl) // Принимает подстроку
str, которую будем искать в документе src с длинной sl.
      {
            initialize(str, src, sl);
      }
      void initialize(char *str, char *src, int sl);

      int find();
};
void low_finding::initialize(char *str, char *src, int sl)
{
      sourcelen=sl;
      source=src; //Сам документ должен где-то хранится пока мы
производим в нём поиск.
      substrlen=strlen(str);
      substr=new char[substrlen+1]; // на один символ больше для
хранения символа конца строки.
      suffix_table=new int[substrlen];
      for(int i=0; i<=substrlen; i++) substr[i]=tolower(str[i]);
      pos=substrlen-1; //По алгоритму сравнение начинается с конца
подстроки.
      build_alphabet();
      build_suffix_table();
}
int low_finding::find()
{
      int i,
            j,
            sbl=substrlen-1;
      unsigned char x;
      for(i=pos; i<sourcelen;)
      {
            x=tolower(source[i]);
            if(x==(unsigned char)substr[sbl])
            {
                  for(j=1; j<substrlen; j++)
                  {
                        if(tolower(source[i-j])!=(unsigned
char)substr[sbl-j])
                        {
                                    i+=suffix_table[j];
                                    break;
                        }
                  }
                  if(j==substrlen)
                  {
                        pos=i+1;
                        return i-sbl;
                  }
```

```
            }
            else i+=alphabet[x];
      }
      pos=sourcelen;
      return -1;
}

//Хэш таблицы
class my_hash_base{
protected:
      unsigned short int *row_len,
            **hash_table,
            *file_count;
      unsigned int len,
            str_len,
            str_val,
            file_count_maxlen,
            id;
      static unsigned int max_all_hash_size,
            used_space;
      char *str;
      wchar_t path[MAX_PATH],
            data_path[MAX_PATH];

      void add_table(unsigned short int sl, unsigned short int f,
unsigned short int pos);
      unsigned int search_base(char *s);
      virtual unsigned int calculate(char *s)=0;

public:
      my_hash_base()
      {
            str=NULL;
            row_len=NULL;
            len=0;
      }

      static void set_max_size(unsigned int sz) { max_all_hash_size=sz;
}
      void add(unsigned int val=1);
      void save();
      void close();
      unsigned int is_exist(wchar_t *p);
      void load(wchar_t *p, bool constant);
      void rebuild(wchar_t *p, unsigned int l);

      ~my_hash_base() {close();}
};
unsigned int my_hash_base::max_all_hash_size=0;
```

```
unsigned int my_hash_base::used_space=0;
void my_hash_base::add(unsigned int val)
{
	if(str==NULL||id!=0||val==0) return;
	unsigned int i,
		bl,
		pos;
	char buf[11];
	wchar_t p[MAX_PATH],
		wbuf[11];
	fstream data;
	wcscpy(p, data_path);
	toa(str_len, buf);
	bl=strlen(buf);
	MultiByteToWideChar(CP_ACP, NULL, buf, bl, wbuf, bl);
	wbuf[bl]=L'\0';
	wcscat(p, wbuf);
	i=file_count_maxlen;
	if(str_len>=file_count_maxlen) endlessmass(file_count, i,
((str_len-i)/1024+1)*1024, file_count_maxlen);
	while(i<file_count_maxlen) file_count[i++]=0;
	toa(file_count[str_len], buf);
	bl=strlen(buf);
	MultiByteToWideChar(CP_ACP, NULL, buf, bl, wbuf, bl);
	wbuf[bl]=L'\0';
	wcscat(p, L"_");
	wcscat(p, wbuf);
	wcscat(p, L".txt");
	data.open(p, ios::out|ios::app|ios::binary);
	if(!data) throw p;
	data.close();
	data.open(p, ios::in|ios::out|ios::ate|ios::binary);
	if(!data) throw p;
	data.seekp(0, ios::end);
	if((data.tellp()/(str_len+7))>=65535)
	{
		data.close();
		file_count[str_len]++;
		p[wcslen(p)-4-wcslen(wbuf)]='\0';
		toa(file_count[str_len], buf);
		bl=strlen(buf);
		MultiByteToWideChar(CP_ACP, NULL, buf, bl, wbuf, bl);
		wbuf[bl]=L'\0';
		wcscat(p, wbuf);
		wcscat(p, L".txt");
		data.open(p, ios::out|ios::app|ios::binary);
		if(!data) throw p;
	}
	data.write(str, str_len);
```

```
	data.write("\t", 1);
	data.write((char *)&val, sizeof(val));
	data.write("\r\n", 2);
	pos=(unsigned short int)(data.tellp()/(str_len+7)-1);
	data.close();
	add_table(str_len, file_count[str_len], (unsigned short int)pos);
	id=val;
}
void my_hash_base::save()
{
	if(row_len==NULL) return;
	unsigned int i,
		j,
		k,
		bl=0;
	unsigned short int *buf;
	wchar_t p[MAX_PATH];
	wcscpy(p, path);
	wcscat(p, L"hash");
	ofstream out(p, ios::binary);
	if(!out) throw p;
	buf=new unsigned short int[67585];
	for(i=0; i<len; i++)
	{
		buf[bl++]=row_len[i];
		for(j=0; j<row_len[i]; j++) buf[bl++]=hash_table[i][j];
		if(bl>=2048)
		{
			for(j=0; j+2048<=bl; j+=2048) out.write((char
*)(buf+j), 2048*sizeof(*buf));
			for(k=0; j<bl; j++, k++) buf[k]=buf[j];
			bl=k;
		}
	}
	out.write((char *)buf, bl*sizeof(*buf));
	delete [] buf;
	out.close();
	wcscpy(p, path);
	wcscat(p, L"properties");
	out.open(p, ios::binary);
	if(!out) throw p;
	out.write((char*)&len, sizeof(len));
	out.write((char*)&file_count_maxlen, sizeof(file_count_maxlen));
	out.write((char*)file_count,
file_count_maxlen*sizeof(*file_count));
	out.close();
}
void my_hash_base::close()
{
```

```
	unsigned int i;
	if(str!=NULL) delete [] str;
	if(row_len!=NULL)
	{
		for(i=0; i<len; i++) if(row_len[i]>0) delete []
hash_table[i];
		delete [] row_len;
		delete [] hash_table;
		delete [] file_count;
	}
	str=NULL;
	row_len=NULL;
}
unsigned int my_hash_base::is_exist(wchar_t *p)
{
	wchar_t pa[MAX_PATH];
	if(_waccess(p, 0)) return 0;
	wcscpy(pa, p);
	if(pa[wcslen(pa)-1]!=L'\\') wcscat(pa, L"\\");
	wcscat(pa, L"properties");
	ifstream in(pa, ios::binary);
	if(!in) return 0;
	in.read((char*)&len, sizeof(len));
	in.close();
	return len;
}
void my_hash_base::load(wchar_t *p, bool constant=0)
{
	unsigned short int *buf;
	unsigned int i,
		j,
		bl=0;
	wchar_t pa[MAX_PATH];
	if(len==0) throw len;
	used_space+=len*(sizeof(hash_table)+sizeof(*row_len));
	if(used_space>max_all_hash_size) throw max_all_hash_size;
	wcscpy(path, p);
	if(path[wcslen(path)-1]!=L'\\') wcscat(path, L"\\");
	wcscpy(data_path, path);
	wcscat(data_path, L"data\\");
	wcscpy(pa, path);
	wcscat(pa, L"properties");
	ifstream in(pa, ios::binary);
	if(!in) throw pa;
	in.seekg(sizeof(len), ios::beg);
	in.read((char*)&file_count_maxlen, sizeof(file_count_maxlen));
	file_count=new unsigned short int[file_count_maxlen];
	in.read((char*)file_count,
file_count_maxlen*sizeof(*file_count));
```

```
	in.close();
	if(!constant)_wremove(pa);
	wcscpy(pa, path);
	wcscat(pa, L"hash");
	in.open(pa, ios::binary);
	if(!in) throw pa;
	hash_table=new unsigned short int *[len];
	row_len=new unsigned short int [len];
	buf=new unsigned short int[2048];
	in.read((char *)buf, 2048*sizeof(*buf));
	for(i=0; i<len; i++)
	{
		if(bl==2048)
		{
			in.read((char *)buf, 2048*sizeof(*buf));
			bl=0;
		}
		row_len[i]=buf[bl++];
		if(row_len[i]>0)
		{
			used_space+=row_len[i]*sizeof(**hash_table);
			if(used_space>max_all_hash_size)
			{
				delete [] buf;
				throw max_all_hash_size;
			}
			hash_table[i]=new unsigned short int[row_len[i]];
			for(j=0; j<row_len[i]; j++)
			{
				if(bl==2048)
				{
					in.read((char *)buf, 2048*sizeof(*buf));
					bl=0;
				}
				hash_table[i][j]=buf[bl++];
			}
		}
	}
	delete [] buf;
	in.close();
}
void my_hash_base::rebuild(wchar_t *p, unsigned int l)
{
	unsigned int sl,
		sl7,
		fc,
		cs,
		pos,
		fnl,
```

```
            i,
            j;
      char *s,
            *f,
            *fl,
            buf[MAX_PATH];
      wchar_t pa[MAX_PATH],
            wbuf[MAX_PATH];
      ifstream in;
      WIN32_FIND_DATA file_data;
      HANDLE hf;
      if(l==0) throw "Длинна хэша должна быть больше 0";
      if(wcslen(p)>230) throw "Слишком длинный путь к папке хэша.";
      len=l;
      used_space+=len*(sizeof(hash_table)+sizeof(*row_len));
      if(used_space>max_all_hash_size) throw max_all_hash_size;
      file_count_maxlen=1024;
      file_count=new unsigned short int [file_count_maxlen];
      for(i=0; i<file_count_maxlen; i++) file_count[i]=0;
      wcscpy(path, p);
      if(path[wcslen(path)-1]!=L'\\') wcscat(path, L"\\");
      crtdir(path);
      wcscpy(data_path, path);
      wcscat(data_path, L"data\\");
      crtdir(data_path);
      hash_table=new unsigned short int *[len];
      row_len=new unsigned short int [len];
      for(i=0; i<len; i++) row_len[i]=0;
      wcscpy(pa, data_path);
      wcscat(pa, L"*");
      hf=FindFirstFile(pa, &file_data);
      if (hf!=INVALID_HANDLE_VALUE)
      {
            do
            {
                  if(file_data.cFileName[0]==L'.') continue;
                  wcscpy(pa, data_path);
                  wcscat(pa, file_data.cFileName);
                  fnl=wcslen(file_data.cFileName);
                  for(i=0; file_data.cFileName[i]!=L'_'&&i<fnl; i++)
wbuf[i]=file_data.cFileName[i];
                  if(i>5||i==fnl||file_data.cFileName[i]!=L'_')
continue;
                  wbuf[i++]=L'\0';
                  WideCharToMultiByte(CP_ACP, NULL, wbuf, i, buf, i,
NULL, NULL);
                  try{
                  ato(buf, sl);
                  }
```

```
                catch(char ex) {continue;}
                if(sl==0||sl>65535) continue;
                for(j=0; file_data.cFileName[i]!=L'.'&&i<fnl; j++,
i++) wbuf[j]=file_data.cFileName[i];
                if(fnl-i!=4||j>5) continue;

     if(file_data.cFileName[i]!=L'.'||file_data.cFileName[i+1]!=L't'||
file_data.cFileName[i+2]!=L'x'||file_data.cFileName[i+3]!=L't')
continue;
                wbuf[j++]=L'\0';
                WideCharToMultiByte(CP_ACP, NULL, wbuf, j, buf, j,
NULL, NULL);
                try{
                ato(buf, fc);
                }
                catch(char ex) {continue;}
                if(fc>65535) continue;
                in.open(pa, ios::binary);
                if(!in) throw pa;
                i=file_count_maxlen;
                if(sl>=file_count_maxlen) endlessmass(file_count,
file_count_maxlen, ((sl-file_count_maxlen)/1024+1)*1024,
file_count_maxlen);
                while(i<file_count_maxlen) file_count[i++]=0;
                if(fc>file_count[sl]) file_count[sl]=fc;
                sl7=sl+7;
                cs=(sl7/4096+1)*4096;
                f=new char[cs+sl7];
                in.read(f, cs);
                fl=f+in.gcount();
                s=f;
                for(pos=0;; pos++)
                {
                     if(s+sl7>fl)
                     {
                          if(in.eof()) break;
                          for(i=0; s<fl; i++, s++) f[i]=*s;
                          fl=f+i;
                          s=f;
                          in.read(fl, cs);
                          fl+=in.gcount();
                     }
                     str_len=sl;
                     str_val=calculate(s);
                     add_table((unsigned short int)sl, (unsigned short
int)fc, (unsigned short int)pos);
                     s+=sl7;
                }
                in.close();
```

```
                delete [] f;
            }
            while (FindNextFile(hf, &file_data)!=0);
            FindClose(hf);
        }
}
void my_hash_base::add_table(unsigned short int sl, unsigned short int
f, unsigned short int pos)
{
      unsigned int rl=row_len[str_val];
      if(rl+3>65535) throw "Превышена максимальная длинна строки хэш
таблицы";
      used_space+=3*sizeof(**hash_table);
      if(used_space>max_all_hash_size) throw max_all_hash_size;
      if(rl==0) hash_table[str_val]=new unsigned short int[3];
      else endlessmass(hash_table[str_val], rl, 3, rl);
      hash_table[str_val][row_len[str_val]++]=sl;
      hash_table[str_val][row_len[str_val]++]=f;
      hash_table[str_val][row_len[str_val]++]=pos;
}
unsigned int my_hash_base::search_base(char *s)
{
      unsigned int i,
            j,
            bl,
            p_sl,
            previus_fl,
            rl;
      ifstream data;
      char buf[11],
            *sbuf;
      wchar_t p[MAX_PATH]=L"",
            wbuf[11];
      if(str_len==0||str_len>65535)
      {
            if(str!=NULL) delete [] str;
            str=NULL;
            throw str_len;
      }
      if(str!=NULL) delete [] str;
      str=new char[str_len+1];
      for(i=0; i<str_len; i++) str[i]=s[i];
      id=0;
      str_val=calculate(str);
      rl=row_len[str_val];
      if(rl==0) return 0;
      else
      {
            sbuf=new char[str_len+1];
```

```
            for(i=0; i<rl; i+=3)
            {
                  if(str_len==hash_table[str_val][i])
                  {
                        if(*p==L'\0')
                        {
                              wcscpy(p, data_path);
                              toa(str_len, buf);
                              bl=strlen(buf);
                              MultiByteToWideChar(CP_ACP, NULL, buf, bl,
wbuf, bl);
                              wbuf[bl]=L'\0';
                              wcscat(p, wbuf);
                              wcscat(p, L"_");
                              p_sl=wcslen(p);
                              previus_fl=hash_table[str_val][i+1]+1;
                        }
                        if(previus_fl!=hash_table[str_val][i+1])
                        {
                              p[p_sl]=L'\0';
                              previus_fl=hash_table[str_val][i+1];
                              toa(previus_fl, buf);
                              bl=strlen(buf);
                              MultiByteToWideChar(CP_ACP, NULL, buf, bl,
wbuf, bl);
                              wbuf[bl]=L'\0';
                              wcscat(p, wbuf);
                              wcscat(p, L".txt");
                              if(data.is_open()) data.close();
                              data.open(p, ios::binary);
                              if(!data) throw p;
                        }
                        data.seekg(hash_table[str_val][i+2]*(str_len+7),
ios::beg);
                        data.read(sbuf, str_len);
                        for(j=0; j<str_len; j++) if(sbuf[j]!=str[j])
break;
                        if(j==str_len)
                        {
                              delete [] sbuf;
                              data.seekg(1, ios::cur);
                              data.read((char *)&id, sizeof(id));
                              data.close();
                              return id;
                        }
                  }
            }
            delete [] sbuf;
            if(data.is_open()) data.close();
```

```
	}
	return 0;
}

class my_hash_str : public my_hash_base{
	unsigned int calculate(char *s);
public:
	my_hash_str():my_hash_base() {};

	unsigned int search(char *s);
};
unsigned int my_hash_str::calculate(char *s)
{
	unsigned int h=0,
		l=str_len*8,
		bit=1,
		pos,
		i,
		j;
	static const unsigned char bit_8[8]={1, 2, 4, 8, 16, 32, 64,
128};
	if(l<=32)
	{
		for(i=0, j=0; i<l; i+=8, j++) h|=(unsigned int)(unsigned
char)s[j]<<i;
		return h%len;
	}
	for(i=1; i<=32; i++)
	{
		pos=((l*i)>>5)-1;
		if(s[pos>>3]&bit_8[pos%8]) h|=bit;
		bit<<=1;
	}
	return h%len;
}
unsigned int my_hash_str::search(char *s)
{
	str_len=strlen(s);
	return search_base(s);
}

class my_hash_int : public my_hash_base{
	unsigned int calculate(char *s);
public:
	my_hash_int():my_hash_base() {};

	unsigned int search(unsigned int *s, unsigned int l);
};
unsigned int my_hash_int::calculate(char *s)
```

```
{
	unsigned int h=0,
		l=str_len<<1,
		bit=1,
		pos,
		i;
	static const unsigned char bit_8[8]={1, 2, 4, 8, 16, 32, 64,
128};
	if(l<=32)
	{
		if(l==8) return (*(unsigned int *)s)%len;
		if(l==16)
		{
			h|=(unsigned int)*(unsigned short int *)(s);
			h|=((unsigned int)*(unsigned short int *)(s+4))<<16;
			return h%len;
		}
		if(l==24)
		{
			h|=(unsigned int)*(short int *)(s);
			h|=((unsigned int)(unsigned char)s[4])<<16;
			h|=((unsigned int)(unsigned char)s[8])<<24;
			return h%len;
		}
		h|=(unsigned int)(unsigned char)s[0];
		h|=((unsigned int)(unsigned char)s[4])<<8;
		h|=((unsigned int)(unsigned char)s[8])<<16;
		h|=((unsigned int)(unsigned char)s[12])<<24;
		return h%len;
	}
	for(i=1; i<=32; i++)
	{
		pos=((l*i)>>5)-1;
		if(s[((pos>>3)<<2)]&bit_8[pos%8]) h|=bit;
		bit<<=1;
	}
	return h%len;
}
unsigned int my_hash_int::search(unsigned int *s, unsigned int l)
{
	str_len=l*4;
	return search_base((char *)s);
}

//Скачивание страниц
class page {
	unsigned int index_num, // Номер страницы оглавления яндекса.
			indexlen; // Длинна страницы оглавления яндекса.
	bool exact_query,
```

```
            save_links,
            xml,
            new_index;
      char *index,
            *serverreq;
      wchar_t start_path[MAX_PATH],
            index_path[MAX_PATH],
            doc_path[MAX_PATH],
            *wreqword,
            *xml_qoptions;
      finding indexfind,
            next_exist,
            links_num_find;

      bool get_page(wchar_t *s, wchar_t *r, wchar_t *p, bool ll);

public:
      page()
      {
            wreqword=NULL;
            index=NULL;
      }
      page(char *w, wchar_t *p, bool e, bool ll, bool xm=0, char
*xmop=NULL)
      {
            initialize(w, p, e, ll, xm, xmop);
      }

      void initialize(char *w, wchar_t *p, bool e, bool ll, bool xm,
char *xmop);
      void get_index_page();
      void get_index_page_xml();
      char *get_serverreq();
      wchar_t *get_new_page(int dn);

      ~page()
      {
            if(index!=NULL) delete [] index;
            if(serverreq!=NULL) delete [] serverreq;
            if(wreqword!=NULL) delete [] wreqword;
            if(xml_qoptions!=NULL) delete [] xml_qoptions;
      }
};
void page::initialize(char *w, wchar_t *p, bool e, bool ll, bool xm=0,
char *xmop=NULL)
{
      unsigned int sl;
      xml=xm;
      xml_qoptions=NULL;
```

```
	if(xml)
	{
		if(xmop==NULL) throw "ОШИБКА: Не заданы опции для запроса
Yandex XML";
		sl=strlen(xmop);
		if(sl==0) throw "ОШИБКА: Не заданы опции для запроса Yandex
XML";
		xml_qoptions=new wchar_t[sl+1];
		MultiByteToWideChar(CP_ACP, NULL, xmop, sl, xml_qoptions,
sl);
		xml_qoptions[sl]=L'\0';
	}
	wcscpy(start_path, p);
	wcscpy(index_path, p);
	wcscat(index_path, L"index\\");
	crtdir(index_path);
	sl=strlen(w);
	wreqword=new wchar_t[sl+1];
	MultiByteToWideChar(CP_ACP, NULL, w, sl, wreqword, sl);
	wreqword[sl]=L'\0';
	index_num=indexlen=0;
	if(xml) indexfind.initialize("<url>http", index, indexlen);
	else
	{
		indexfind.initialize("<a class=\"b-link serp-item__title-
link\" href=\"http", index, indexlen);
		next_exist.initialize("<span
class=\"button__text\">РЎР»РµРґСѓСЋС‰Р°СЏ</span></a></span>", index,
indexlen);
		links_num_find.initialize("РџPsСЃPjPsC,СЂРµC,СЊ PI
РґСЂСѓPiPёC… PïPsPёСЃPєPsPIC‹C… СЃPёСЃC,РµPjP°C…", index, indexlen);
	}
	index=NULL;
	exact_query=e;
	save_links=ll;
	serverreq=NULL;
	if(xml) get_index_page_xml();
	else get_index_page();
}
bool page::get_page(wchar_t *s, wchar_t *r, wchar_t *p, bool ll)
{
	bool ok=0;
	// инициализируем WinInet
	HINTERNET hInternet=::InternetOpen(TEXT("Mozilla/5.0 (Windows NT
6.1; WOW64; rv:38.0) Gecko/20100101 Firefox/38.0"),
INTERNET_OPEN_TYPE_PRECONFIG, NULL, NULL, 0);
	if(hInternet!=NULL)
	{
		// открываем HTTP сессию
```

```
          HINTERNET hConnect=::InternetConnect(hInternet, s,
INTERNET_DEFAULT_HTTP_PORT, NULL, NULL, INTERNET_SERVICE_HTTP, 0, 1u);
         if(hConnect!=NULL)
          {
               // открываем запрос
               HINTERNET hRequest=::HttpOpenRequest(hConnect,
TEXT("GET"), r, NULL, NULL, 0, INTERNET_FLAG_KEEP_CONNECTION, 1);
               if (hRequest!=NULL)
               {
                    // посылаем запрос
                    BOOL bSend=::HttpSendRequest(hRequest, NULL, 0,
NULL, 0);
                    if (bSend)
                    {
                         // создаём выходной файл
                         ofstream f(p, ios::out|ios::binary);
                         if(!f) throw p;
                         if(ll)
                         {
                              f.write("<selflink>", 10);
                              f.write(serverreq, strlen(serverreq));
                              f.write("</selflink>" , 11);
                         }
                         ok=true;
                         while(true)
                         {
                              // читаем данные
                              char szData[1024];
                              DWORD dwBytesRead;
                              BOOL
bRead=::InternetReadFile(hRequest, szData,sizeof(szData)-1,
&dwBytesRead);
                              // выход из цикла при ошибке или
завершении
                              if(bRead==FALSE||dwBytesRead==0)break;
                              // сохраняем результат
                              f.write(szData, dwBytesRead);
                         }
                         f.close();
                    }
                    // закрываем запрос
                    ::InternetCloseHandle(hRequest);
               }
               // закрываем сессию
               ::InternetCloseHandle(hConnect);
          }
          // закрываем WinInet
          ::InternetCloseHandle(hInternet);
     }
```

```
	return ok;
}
void page::get_index_page()
{
	if(wreqword==NULL) throw "Класс скачивания страниц не
инициализирован.";
	int sleep_len=1;
	wchar_t req[450]=L"/yandsearch?",
		path[MAX_PATH],
		itoa_buf[11];
	_itow(index_num, itoa_buf, 10);
	if(index_num>0)
	{
		wcscat(req, L"p=");
		wcscat(req, itoa_buf);
		wcscat(req, L"&");
	}
	wcscat(req, L"text=");
	wcscat(req, wreqword);
	if(exact_query) wcscat(req, L"&noreask=1");
	wcscpy(path, index_path);
	wcscat(path, itoa_buf);
	wcscat(path, L".html");
	while(true)
	{
		if(!get_page(L"www.yandex.ru", req, path, 0)) throw
"Неудаётся загрузить страницу яндекса.";
		inmass(path, index, indexlen);
		links_num_find.new_source(index, indexlen);
		if(links_num_find.find()==-1)
		{
			cout << "Бан от яндекса. Ждём " << sleep_len << "
секунд.\n";
			Sleep(sleep_len*1000);
			sleep_len++;
		}
		else
		{
			index_num++;
			indexfind.new_source(index, indexlen);
			next_exist.new_source(index, indexlen);
			return;
		}
	}
}
void page::get_index_page_xml()
{
	if(wreqword==NULL) throw "Класс скачивания страниц не
инициализирован.";
```

```
	wchar_t req[4096]=L"/search/xml?",
		path[MAX_PATH],
		itoa_buf[11];
	wcscat(req, L"query=");
	wcscat(req, wreqword);
	_itow(index_num, itoa_buf, 10);
	if(index_num>0)
	{
		wcscat(req, L"&page=");
		wcscat(req, itoa_buf);
	}
	wcscat(req, xml_qoptions);
	if(exact_query) wcscat(req, L"&noreask=1");
	wcscpy(path, index_path);
	wcscat(path, itoa_buf);
	wcscat(path, L".xml");
	if(!get_page(L"www.yandex.ru", req, path, 0)) throw "Неудаётся
загрузить страницу яндекса.";
	inmass(path, index, indexlen);
	new_index=1;
	index_num++;
	indexfind.new_source(index, indexlen);
	return;
}
char *page::get_serverreq()
{
	if(index==NULL) throw "Класс скачивания страниц не
инициализирован.";
	int start;
	unsigned int i;
	while(true)
	{
		start=indexfind.find();
		if(start==-1)
		{
			if(xml)
			{
				if(!new_index)
				{
					delete [] index;
					get_index_page_xml();
				}
				else return "\0";
			}
			else
			{
				if(next_exist.find()!=-1)
				{
					delete [] index;
```

```
                        get_index_page();
                    }
                    else return "\0";
                }
            }
            else
            {
                new_index=0;
                if(xml) start+=9;
                else start+=50;
                if(index[start]=='s') start++;
                start+=3;
                for(i=start; index[i]!='\"'&&index[i]!='<';) i++;
                if(serverreq!=NULL) delete [] serverreq;
                serverreq=new char[i-start+1];
                for(i=0; index[start]!='\"'&&index[start]!='<';
start++, i++)
                {
                    serverreq[i]=tolower(index[start]);
                }
                serverreq[i]='\0';
                return serverreq;
            }
        }
}
wchar_t *page::get_new_page(int dn)
{
        if(serverreq[0]=='\0') return L"\0";
        unsigned int sr_len=strlen(serverreq),
            s_len,
            r_len,
            i,
            j;
        bool ok=0;
        char *req,
            *server;
        wchar_t itow_buf[20],
            *wreq,
            *wserver;
        req=new char[sr_len+1];
        server=new char[sr_len+1];
        for(i=0; serverreq[i]!='/'&&i<sr_len; i++)
server[i]=serverreq[i];
        server[i]='\0';
        for(j=0; i<sr_len; j++, i++) req[j]=serverreq[i];
        req[j]='\0';
        wcscpy(doc_path, start_path);
        _itow(dn, itow_buf, 10);
        wcscat(doc_path, itow_buf);
```

```
	wcscat(doc_path, L".html");
	s_len=strlen(server);
	wserver=new wchar_t[s_len+1];
	MultiByteToWideChar(CP_ACP, NULL, server, s_len, wserver, s_len);
	wserver[s_len]='\0';
	r_len=strlen(req);
	wreq=new wchar_t[r_len+1];
	MultiByteToWideChar(CP_ACP, NULL, req, r_len, wreq, r_len);
	wreq[r_len]='\0';
	ok=get_page(wserver, wreq, doc_path, save_links);
	delete [] req;
	delete [] server;
	delete [] wreq;
	delete [] wserver;
	if(ok) return doc_path;
	else return L"\0";
}

//Раскодирование
class encoding {
public:
	char utf8_buf[4]; // Буфер, который хранит строку  восьмибитных
символов, обозначающих символ utf8.
	static unsigned int utf8_cp1251_table[128]; // Таблица, которая в
ячейке с номером, соответствующем номеру символа в cp1251, хранит код
этого символа в utf8.

	char utf8_to_cp1251(char *m, unsigned int &p, char
missing_symbol=-104); // Принимает указатель на строку символов, и
позицию первого байта символа utf8 и возвращает соответствующий ему
символ cp1251, переводит указатель на следующий символ.
	char *cp1251_to_utf8(char ch); // Принимает символ cp1251 и
возвращает строку обозначающую символ utf8.
};
unsigned int encoding::utf8_cp1251_table[128] = {0xd082, 0xd083,
0xe2809a, 0xd193, 0xe2809e, 0xe280a6, 0xe280a0, 0xe280a1, 0xe282ac,
0xe280b0, 0xd089, 0xe280b9, 0xd08a, 0xd08c, 0xd08b, 0xd08f,
                                                        0xd192,
0xe28098, 0xe28099, 0xe2809c, 0xe2809d, 0xe280a2, 0xe28093, 0xe28094,
0x0, 0xe284a2, 0xd199, 0xe280ba, 0xd19a, 0xd19c, 0xd19b, 0xd19f,
                                                        0xc2a0,
0xd08e, 0xd19e, 0xd088, 0xc2a4, 0xd089, 0xc2a6, 0xc2a7, 0xd081,
0xc2a9, 0xd084, 0xc2ab, 0xc2ac, 0xc2ad, 0xc2ae, 0xd087,
                                                        0xc2b0,
0xc2b1, 0xd086, 0xd196, 0xd291, 0xc2b5, 0xc2b6, 0xc2b7, 0xd191,
0xe28496, 0xd194, 0xc2bb, 0xd198, 0xd085, 0xd195, 0xd197,
                                                        0xd090,
0xd091, 0xd092, 0xd093, 0xd094, 0xd095, 0xd096, 0xd097, 0xd098,
0xd099, 0xd09a, 0xd09b, 0xd09c, 0xd09d, 0xd09e, 0xd09f,
```

```
                                                                0xd0a0,
0xd0a1, 0xd0a2, 0xd0a3, 0xd0a4, 0xd0a5, 0xd0a6, 0xd0a7, 0xd0a8,
0xd0a9, 0xd0aa, 0xd0ab, 0xd0ac, 0xd0ad, 0xd0ae, 0xd0af,
                                                                0xd0b0,
0xd0b1, 0xd0b2, 0xd0b3, 0xd0b4, 0xd0b5, 0xd0b6, 0xd0b7, 0xd0b8,
0xd0b9, 0xd0ba, 0xd0bb, 0xd0bc, 0xd0bd, 0xd0be, 0xd0bf,
                                                                0xd180,
0xd181, 0xd182, 0xd183, 0xd184, 0xd185, 0xd186, 0xd187, 0xd188,
0xd189, 0xd18a, 0xd18b, 0xd18c, 0xd18d, 0xd18e, 0xd18f};

char encoding::utf8_to_cp1251(char *m, unsigned int &p, char
missing_symbol)
{
      // Если в cp1251 нет соответствующего символа то возвращаем
значение 0x98, которому не соответвтвует никакой символ.
      unsigned int buf,
            i;
      if((unsigned char)m[p]<0x80) // Символы ANSI в юникоде
записываются так же, поэтому если значение первого байта меньше 128,
то просто возвращаем его.
      {
            return m[p++];
      }
      // Если символ юникода состоит более чем из 1 байта, то первый
байт начинается, со стольких единиц, сколькими байтами кодируется
символ, затем идёт 0 и значащие символы.
      if((unsigned char)m[p]<0xe0) // Для двухбайтного символа третья
цифра 0 следовательно его максимальное значение 0xcf.
      {
            if((m[p+1]&0xc0)!=0x80) return m[p++];
            buf=(((unsigned char)m[p])<<8)+((unsigned char)m[p+1]);
            p+=2;
            for(i=0; i<128; i++) if(buf==utf8_cp1251_table[i]) return
(char)(i+128);
            if(i==128) return missing_symbol;
      }
      if((unsigned char)m[p]<0xf0) // Если символ 3 байтный, то
четвёртая цифра 0 следовательно максимальное значение 0xef.
      {
            if((m[p+1]&0xc0)!=0x80) return m[p++];
            if((m[p+2]&0xc0)!=0x80) return m[p++];
            buf=(((unsigned char)m[p])<<16)+(((unsigned
char)m[p+1])<<8)+((unsigned char)m[p+2]);
            p+=3;
            for(i=0; i<128; i++) if(buf==utf8_cp1251_table[i]) return
(char)(i+128);
            return missing_symbol;
      }
```

```
	// Теперь мы знаем, что символ содержит более 3 байт, и для него
нет аналога в cp1251.
	if((m[p+1]&0xc0)!=0x80) return m[p++];
	if((m[p+2]&0xc0)!=0x80) return m[p++];
	if((m[p+3]&0xc0)!=0x80) return m[p++];
	p+=4;
	return missing_symbol;
}
char *encoding::cp1251_to_utf8(char ch)
{
	unsigned int buf, i=0;
	if((unsigned char)ch<128) // Символы ANSI в utf8 записываются так
же.
	{
		utf8_buf[0]=ch;
		utf8_buf[1]='\0';
		return utf8_buf;
	}
	buf=utf8_cp1251_table[(unsigned char)ch-128]; // Записываем в
буфер значение из таблицы далее разделяем по байтам и записываем в
строку вывода.
	if((int)(buf/0x10000)>0) utf8_buf[i++]=(char)(buf/0x10000);
	utf8_buf[i++]=(char)(buf/0x100);
	utf8_buf[i++]=(char)(buf%0x100);
	utf8_buf[i]='\0';
	return utf8_buf;
}

//Обработка страниц
class scaning {
	unsigned int doc_pos,
		doc_len,
		s_pos,
		sp_pos,
		min_length,
		min_words;
	char *doc,
		*sentence,
		missing_symbol;
	bool data_write,
		save_open_tag,
		in_sentence,
		space;
	my_hash_str amp_hash;

	int def_code(char *w);
	void tag(unsigned int &k);
public:
	scaning()
```

```
	{
		min_length=0;
	}
	scaning(unsigned int ml, unsigned int mw, bool sot, char ms)
	{
		initialize(ml, mw, sot, ms);
	}
	void initialize(unsigned int ml, unsigned int mw, bool sot, char
ms);

	unsigned int scan(wchar_t *path, char *word, bool wr=0);

	~scaning()
	{
		if(min_length!=0) amp_hash.close();
	}
};
void scaning::initialize(unsigned int ml, unsigned int mw, bool sot,
char ms)
{
	min_length=ml+1;
	min_words=mw;
	save_open_tag=sot;
	missing_symbol=ms;
	amp_hash.is_exist(L"amp_hash\\");
	amp_hash.load(L"amp_hash\\", true);
}
int scaning::def_code(char *w)
{
	int meta_start,
		c_pos;
	unsigned int i,
		j,
		wl=strlen(w),
		meta_len=0;
	char *word_cp1251,
		*meta=NULL;
	encoding enc;
	static low_finding m("<meta", doc, doc_len);
	static low_finding c("charset", meta, meta_len);
	static low_finding cp("windows-1251", meta, meta_len);
	static low_finding u("utf-8", meta, meta_len);
	m.new_source(doc, doc_len);
	for(meta_start=m.find(); meta_start!=-1; meta_start=m.find())
	{
		for(meta_len=meta_start;
doc[meta_len]!='>'&&meta_len<doc_len;) meta_len++;
		meta_start+=5;
		meta_len-=meta_start;
```

```
            if(meta_len<14) continue;
            meta=doc+meta_start;
            c.new_source(meta, meta_len);
            c_pos=c.find();
            if(c_pos!=-1)
            {
                  c_pos+=7;
                  cp.new_source(meta, meta_len);
                  cp.set_position(c_pos);
                  if(cp.find()!=-1) return 1;
                  u.new_source(meta, meta_len);
                  u.set_position(c_pos);
                  if(u.find()!=-1)
                  {
                        for(j=0, i=0; i<doc_len; j++)
doc[j]=enc.utf8_to_cp1251(doc, i, missing_symbol);
                        doc_len=j;
                        return 1;
                  }
            }
      }
      word_cp1251=new char[wl+1];
      for(i=0; i<wl;)
      {
            j=0;
            while(i<wl)
            {

      if(((w[i]>='А')&&(w[i]<='я'))||(w[i]=='ё')||(w[i]=='Ё'))
word_cp1251[j++]=w[i++];
                  else
                  {
                        i++;
                        break;
                  }
            }
            if(j<3) continue;
            word_cp1251[j]='\0';
            low_finding fword_cp1251(word_cp1251, doc, doc_len);
            if(fword_cp1251.find()!=-1)
            {
                  delete [] word_cp1251;
                  return 1;
            }
      }
      for(j=0, i=0; i<doc_len; j++) doc[j]=enc.utf8_to_cp1251(doc, i,
missing_symbol);
      doc_len=j;
      for(i=0; i<wl;)
```

```
	{
		j=0;
		while(i<wl)
		{

	if(((w[i]>='A')&&(w[i]<='я'))||(w[i]=='ё')||(w[i]=='Ё'))
word_cp1251[j++]=w[i++];
			else
			{
				i++;
				break;
			}
		}
		if(j<3) continue;
		word_cp1251[j]='\0';
		low_finding fword_cp1251(word_cp1251, doc, doc_len);
		if(fword_cp1251.find()!=-1)
		{
			delete [] word_cp1251;
			return 1;
		}
	}
	delete [] word_cp1251;
	return 0;
}
unsigned int scaning::scan(wchar_t *path, char *word, bool wr)
{
	if(!min_length) throw "Класс сканирования предложений не
инициализирован.";
	unsigned int i,
		j,
		k,
		words,
		sentence_count,
		amp_val,
		tag_len;
	char amp[15];
	unsigned char ch;
	wchar_t opath[MAX_PATH];
	if(path[0]=='\0'||wcslen(path)>=260) return 0;
	inmass(path, doc, doc_len, 1);
	doc[doc_len]='\0';
	if(!wr) if(!def_code(word)) return 0;
	doc[doc_len]='\0';
	sentence_count=s_pos=sp_pos=doc_pos=0;
	space=in_sentence=false;
	data_write=wr;
	while(doc_pos<doc_len)
	{
```

```
            if(doc[doc_pos]=='<') tag(doc_pos);
            else if(data_write)
            {
                  if(doc[doc_pos]=='&')
                  {
                        i=doc_pos+1;
                        k=i+12;
                        j=0;
                        while(i<doc_len&&i<k)
                        {
                              if(doc[i]=='\r'||doc[i]=='\n'||doc[i]=='
'||doc[i]=='\t'||doc[i]=='\v'||doc[i]==-96||doc[i]==';') break;
                              amp[j++]=doc[i++];
                        }
                        if(doc[i]==';'&&j>0)
                        {
                              doc_pos=i;
                              if(j==2)
if((*amp=='l'||*amp=='g')&&amp[1]=='t') doc_pos-=(1+j);
                              if(doc_pos==i)
                              {
                                    amp[j]='\0';
                                    amp_val=amp_hash.search(amp);
                                    if(amp_val==0)
doc[doc_pos]=missing_symbol;
                                    else doc[doc_pos]=amp_val;
                              }
                        }
                  }

      if(doc[doc_pos]=='\r'||doc[doc_pos]=='\n'||doc[doc_pos]=='
'||doc[doc_pos]=='\t'||doc[doc_pos]=='\v'||doc[doc_pos]==-96)
                  {
                        if(in_sentence) space=true;
                        doc_pos++;
                  }
                  else
                  {
                        in_sentence=true;
                        if(space)
                        {
                              doc[s_pos++]=' ';
                              space=false;
                        }
                        doc[s_pos++]=doc[doc_pos++];
                  }
            }
            else doc_pos++;
      }
```

```
	doc[s_pos]='\0';
	doc_len=s_pos;
	s_pos=doc_pos=0;
	sentence=new char[doc_len<<1];
	while(doc_pos<doc_len)
	{
		if(doc[doc_pos]=='<')
		{
			s_pos=sp_pos;
			if(save_open_tag)
			{
				while(doc[doc_pos]!='>')
sentence[s_pos++]=doc[doc_pos++];
				sentence[s_pos++]='>';
				sentence[s_pos++]='\r';
				sentence[s_pos++]='\n';
			}
			else while(doc[doc_pos++]!='>');
			doc_pos++;
			tag_len=s_pos-sp_pos;
		}
		else
if(doc[doc_pos]=='.'||doc[doc_pos]=='?'||doc[doc_pos]=='!')
		{
			sentence[s_pos++]=doc[doc_pos++];
			if(doc_pos>=doc_len)
			{
				ch=sentence[sp_pos+tag_len];
				if((ch>64&&ch<91)||(ch>191&&ch<224)||ch==168)
				{
					sentence[s_pos++]='\r';
					sentence[s_pos++]='\n';
					sp_pos=s_pos;
				}
				break;
			}
			if(doc[doc_pos]!=' '&&doc[doc_pos]!='<') continue;
			if(doc[doc_pos-1]=='.'&&doc[doc_pos]==' ')
			{
				ch=doc[doc_pos+1];
				if(!((ch>64&&ch<91)||(ch>191&&ch<224)||ch==168))
continue;
				if(doc[doc_pos-3]==' '&&isupper((unsigned
char)doc[doc_pos-2])) continue;
			}
			if(doc[doc_pos]==' ') doc_pos++;
			ch=sentence[sp_pos+tag_len];
```

```
	if(!((ch>64&&ch<91)||(ch>191&&ch<224)||ch==168||ch==150||ch==151|
|ch=='-'))
			{
				s_pos=sp_pos+tag_len;
				continue;
			}
			words=1;
			for(i=sp_pos+tag_len; i<s_pos; i++) if(sentence[i]==' 
') words++;
			if(words<min_words||s_pos-sp_pos-tag_len<min_length)
			{
				s_pos=sp_pos+tag_len;
				continue;
			}
			tag_len=0;
			sentence[s_pos++]='\r';
			sentence[s_pos++]='\n';
			sentence_count++;
			sp_pos=s_pos;
		}
		else sentence[s_pos++]=doc[doc_pos++];
	}
	if(sp_pos>0)
	{
		wcscpy(opath, path);
		opath[wcslen(opath)-4]=L'\0';
		if(wr) wcscat(opath, L"_scan.txt");
		else wcscat(opath, L"txt");
		outmass(opath, sentence, sp_pos);
	}
	delete [] doc;
	delete [] sentence;
	return sentence_count;
}
void scaning::tag(unsigned int &i)
{
	unsigned int k,
		j,
		q,
		s;
	static const char *skip_tag[36]={"a", "abbr", "acronym", "b", 
"bdi", "bdo", "big", "blink", "center", "cite",
"code", "del", "dfn", 
"em", "font", "i", "img", "ins", "kbd", "mark",
"nobr", 
"plaintext", "q", "s", "small", "span", "strike", "strong", "sub", 
"sup",
```

```
                                                                  "time",
"tt", "u", "var", "wbr", "xmp"},
            *open_tag[19]={"address", "article", "blockquote", "dd",
"details", "div", "dt", "footer", "h1", "h2",
                                    "h3", "h4", "h5", "h6", "li", "main",
"p", "td", "th"},
            *continue_tag[1]={"br"};
      i++;
      if(i+2<doc_len)
      {
            if(doc[i]=='!'&&doc[i+1]=='-'&&doc[i+2]=='-')
            {
                  i+=3;
                  while(i+2<doc_len)
                  {
                        if(doc[i]=='-'&&doc[i+1]=='-'&&doc[i+2]=='>')
break;
                        i++;
                  }
                  i+=3;
                  return;
            }
      }
      if(i+6<doc_len)
      {
            if((char)tolower(doc[i])=='s')
if((char)tolower(doc[i+1])=='c'&&(char)tolower(doc[i+2])=='r'&&

      (char)tolower(doc[i+3])=='i'&&(char)tolower(doc[i+4])=='p'&&(char
)tolower(doc[i+5])=='t'&&
                  ((char)tolower(doc[i+6])=='
'||(char)tolower(doc[i+6])=='>'))
            {
                  i+=6;
                  while(i+7<doc_len)
                  {
                        if(doc[i]=='/')
if((char)tolower(doc[i+1])=='s'&&(char)tolower(doc[i+2])=='c'&&(char)t
olower(doc[i+3])=='r'&&

      (char)tolower(doc[i+4])=='i'&&(char)tolower(doc[i+5])=='p'&&(char
)tolower(doc[i+6])=='t'&&doc[i+7]=='>') break;
                        i++;
                  }
                  i+=8;
                  data_write=space=in_sentence=false;
                  return;
            }
      }
```

```
      s=doc[i]=='/'?++i:i;
      while(i<doc_len) if(doc[i++]=='>') break;
      if(i>=doc_len) return;
      for(j=0; j<36; j++)
      {
            for(q=0, k=s; skip_tag[j][q]!='\0'; q++, k++)
if(tolower(doc[k])!=(unsigned char)skip_tag[j][q]) break;
            if((doc[k]=='
'||doc[k]=='>'||doc[k]=='/')&&skip_tag[j][q]=='\0') return;
      }
      space=in_sentence=false;
      for(j=0; j<19; j++)
      {
            for(q=0, k=s; open_tag[j][q]!='\0'; q++, k++)
if(tolower(doc[k])!=(unsigned char)open_tag[j][q]) break;
            if((doc[k]=='
'||doc[k]=='>'||doc[k]=='/')&&open_tag[j][q]=='\0')
            {
                  if(doc[s-1]=='/') data_write=false;
                  else
                  {
                        data_write=true;
                        doc[s_pos++]='<';
                        for(q=0; open_tag[j][q]!='\0'; q++)
doc[s_pos++]=open_tag[j][q];
                        doc[s_pos++]='>';
                  }
                  return;
            }
      }
      if(doc[s-1]!='/')
      {
            for(j=0; j<1; j++)
            {
                  for(q=0, k=s; continue_tag[j][q]!='\0'; q++, k++)
if(tolower(doc[k])!=(unsigned char)continue_tag[j][q]) break;
                  if((doc[k]=='
'||doc[k]=='>'||doc[k]=='/')&&continue_tag[j][q]=='\0')
                  {
                        if(data_write)
                        {
                              doc[s_pos++]='<';
                              for(q=0; continue_tag[j][q]!='\0'; q++)
doc[s_pos++]=continue_tag[j][q];
                              doc[s_pos++]='>';
                        }
                        return;
                  }
            }
```

```
	}
	data_write=false;
}

//Лематизация
class lemma
{
	bool isloaded;
	unsigned int massin_len,
		massout_len;
	char *massin,
		*massout;
	wchar_t opath[MAX_PATH];
	HLEM hEngine;

	void razd();
public:
	lemma() {isloaded=0;}
	void initialize()
	{
		isloaded=1;
		hEngine=sol_LoadLemmatizatorA(
"RussianGrammaticalDictionary\\bin-windows\\lemmatizer.db",
LEME_FASTEST);
	}

	wchar_t *lem(wchar_t *p);

	~lemma()
	{
		if(isloaded)sol_DeleteLemmatizator(hEngine);
	}
};
void lemma::razd()
{
	int i,
		j;
	bool in_word=false;
	massout=new char[massin_len<<1];
	massout_len=0;
	for(i=0; i<(int)massin_len; i++)
	{
		if(massin[i]=='\r'||massin[i]=='\n'||massin[i]==' ')
		{
			in_word=false;
			massout[massout_len++]=massin[i];
		}
		else if(massin[i]>=0&&massin[i]<32) continue;
```

```
            else
if((massin[i]=='\"'||massin[i]=='('||massin[i]=='['||massin[i]=='{'||m
assin[i]=='|'||massin[i]==-117||massin[i]==-111||
                  massin[i]==-109||massin[i]==-89||massin[i]==-
85)&&!in_word)
            {
                  massout[massout_len++]=massin[i];
                  if(massin[i+1]!=' ') massout[massout_len++]=' ';
            }
            else if((unsigned char)massin[i]==152)
            {
                  if(massout_len>0)
                  {
                        if((massout[massout_len-1]==
'||massout[massout_len-1]=='\n')&&massin[i+1]==' ') i++;
                  }
                  else if(massin[i+1]==' ') i++;
            }
            else
            {
                  in_word=true;
                  massout[massout_len++]=massin[i];
            }
      }
      delete [] massin;
      massin=new char[massin_len<<1];
      for(i=massout_len-1, massin_len=0; i>=0; i--)
      {
            if(massout[i]==']'&&!in_word)
            {
                  for(j=i-1; j>=0; j--) if(!isdigit((unsigned
char)massout[j])) break;
                  if(massout[j]=='['&&j!=i-1)
                  {
                        massin[massin_len++]=']';
                        massin[massin_len++]=' ';
                        i--;
                        while(i>j) massin[massin_len++]=massout[i--];
                        massin[massin_len++]=' ';
                        massin[massin_len++]='[';
                        i--;
                        if(i>=0) if(massout[i]!='\n'&&massout[i]!=' ')
massin[massin_len++]=' ';
                  }
            }
            if(massout[i]=='\r'||massout[i]=='\n'||massout[i]==' ')
            {
                  in_word=false;
                  massin[massin_len++]=massout[i];
```

```
            }
            else
if((massout[i]=='!'||massout[i]=='\"'||massout[i]==')'||massout[i]=='*
'||massout[i]==','||massout[i]=='.'||massout[i]==':'||

      massout[i]==';'||massout[i]=='?'||massout[i]==']'||massout[i]=='|
'||massout[i]=='}'||massout[i]==-123||massout[i]==-110||
                  massout[i]==-108||massout[i]==-103||massout[i]==-
101||massout[i]==-87||massout[i]==-82||massout[i]==-69)&&!in_word)
            {
                  massin[massin_len++]=massout[i];
                  if(massout[i]=='.')
                  {
                        if(massout[i+1]=='\r'&&massout[i-1]!=' ')
massin[massin_len++]=' ';
                  }
                  else if(i-1>=0) if(massout[i-1]!='\n'&&massout[i-1]!='
') massin[massin_len++]=' ';
            }
            else
            {
                  in_word=true;
                  massin[massin_len++]=massout[i];
            }
      }
      *massout='\0';
      massout_len=0;
      massin[massin_len]='\0';
      revers(massin);
}
wchar_t *lemma::lem(wchar_t *p)
{
      if(*p=='\0'||wcslen(p)>=260) return L"\0";
      if(!isloaded) return L"\0";
      unsigned int nlemma,
            i,
            j,
            wl;
      char *sentence,
            *word_out;
      wcscpy(opath, p);
      opath[wcslen(opath)-4]=L'\0';
      wcscat(opath, L"_lem.txt");
      inmass(p, massin, massin_len, 1);
      wl=massin_len<<1;
      massin[massin_len]='\0';
      razd();
      for(i=0; i<massin_len;)
      {
```

```
            sentence=massin+i;
            while(massin[i]!='\r') i++;
            massin[i]='\0';
            i+=2;
            HLEMMAS hList=sol_LemmatizePhraseA(hEngine, sentence, 0,
32);
            if(hList!=NULL)
            {
                  nlemma=sol_CountLemmas(hList);
                  for(j=0; j<nlemma; j++)
                  {
                        word_out=massout+massout_len;
                        sol_GetLemmaStringA(hList, j, word_out, wl-
massout_len);
                        massout_len+=strlen(word_out);
                        massout[massout_len++]=' ';
                  }
                  if(j)
                  {
                        massout[massout_len-1]='\r';
                        massout[massout_len++]='\n';
                  }
                  sol_DeleteLemmas(hList);
            }
      }
      outmass(opath, massout, massout_len);
      delete [] massin;
      delete [] massout;
      return opath;
}

//Кодирование
class coding{
      bool use_sentence_hash,
            use_sentence_sort;
      unsigned short int *vocabulary_count;
      unsigned int vocabulary_len,
            vocabulary_maxlen,
            sentence_num;
      static const unsigned int vocabulary_ext=100000;
      char **vocabulary;
      wchar_t vocabulary_path[MAX_PATH];
      my_hash_str word_hash;
      my_hash_int sentence_hash;
public:
      coding()
      {
            vocabulary=NULL;
            use_sentence_hash=0;
```

```
	}
	coding(wchar_t *p, unsigned int whl, unsigned int w, bool uss=1,
unsigned int shl=0, unsigned int s=0, bool ush=0)
	{
		initialize(p, whl, w, uss, shl, s, ush);
	}
	void initialize(wchar_t *p, unsigned int whl, unsigned int w,
bool uss=1, unsigned int shl=0, unsigned int s=0, bool ush=0);

	unsigned int get_sentence_num() {return sentence_num;}
	unsigned int get_words_num() {return vocabulary_len;}
	void code_page(wchar_t *p, bool freq_calc=1);

	~coding();
};
void coding::initialize(wchar_t *p, unsigned int whl, unsigned int w,
bool uss, unsigned int shl, unsigned int s, bool ush)
{
	unsigned int i,
		j,
		vf_len;
	char *word,
		*vf;
	wchar_t buf_path[MAX_PATH];
	wcscpy(vocabulary_path, p);
	wcscpy(buf_path, vocabulary_path);
	wcscat(buf_path, L"word_hash\\");
	if(!(i=whl))
	{
		i=1;
		while(i<w) i<<=1;
		i--;
	}
	if(word_hash.is_exist(buf_path)==i) word_hash.load(buf_path);
	else word_hash.rebuild(buf_path, i);
	use_sentence_sort=uss;
	use_sentence_hash=ush;
	if(ush)
	{
		wcscpy(buf_path, vocabulary_path);
		wcscat(buf_path, L"sentence_hash\\");
		if(!(i=shl))
		{
			i=1;
			while(i<s) i<<=1;
			i--;
		}
		if(sentence_hash.is_exist(buf_path)==i)
sentence_hash.load(buf_path);
```

```
			else sentence_hash.rebuild(buf_path, i);
		}
		wcscpy(buf_path, p);
		wcscat(buf_path, L"vocabulary.txt");
		vocabulary_len=0;
		if(!_waccess(buf_path, 0))
		{
			inmass(buf_path, vf, vf_len);
			_wremove(buf_path);
			word=vf;
			for(i=0; vf[i]!='\t'; i++);
			vf[i]='\0';
			i++;
			ato(word, vocabulary_maxlen);
			word=vf+i;
			while(vf[i]!='\r') i++;
			vf[i]='\0';
			i+=2;
			ato(word, sentence_num);
			vocabulary=new char*[vocabulary_maxlen];
			vocabulary_count=new unsigned short int[vocabulary_maxlen];
			for(vocabulary_len=0; i<vf_len;)
			{
				word=vf+i;
				while(vf[i]!='\t') i++;
				vf[i]='\0';
				ato(word, j);
				vocabulary_count[++vocabulary_len]=(unsigned short
int)j;
				i++;
				word=vf+i;
				while(vf[i]!='\r') i++;
				vf[i]='\0';
				j=vf+i-word+1;
				vocabulary[vocabulary_len]=new char[j];
				strcpy(vocabulary[vocabulary_len], word);
				i+=2;
			}
			delete [] vf;
		}
		else
		{
			sentence_num=0;
			vocabulary_maxlen=vocabulary_ext;
			vocabulary=new char*[vocabulary_maxlen];
			vocabulary_count=new unsigned short int[vocabulary_maxlen];
		}
}
void coding::code_page(wchar_t *p, bool freq_calc)
```

```
{
	if(vocabulary==NULL) throw "Словарь не инициализирован.";
	if(*p=='\0'||wcslen(p)>=260) return;
	unsigned int page_len,
		i,
		j,
		k,
		*int_page,
		*int_sentence,
		int_sentence_len=0;
	char *page,
		*word;
	wchar_t opath[MAX_PATH];
	wcscpy(opath, p);
	opath[wcslen(opath)-7]='\0';
	wcscat(opath, L"code.txt");
	inmass(p, page, page_len, 1);
	page[page_len]='\0';
	int_page=new unsigned int[page_len>>1];
	int_sentence=int_page;
	for(i=0; i<page_len; i++)
	{
		if(page[i]=='<')
		{
			while(page[i]!='>') i++;
			i+=2;
		}
		else if(page[i]=='\n')
		{

	if((freq_calc&&(int_sentence_len>=2))||((!freq_calc)&&(int_senten
ce_len>0)))
			{
				if(use_sentence_sort) qsort(int_sentence,
int_sentence_len, sizeof(int), compare);
				if(use_sentence_hash)
j=sentence_hash.search(int_sentence, int_sentence_len);
				if(j==0||!use_sentence_hash)
				{
					if(freq_calc) sentence_num++;
					if(use_sentence_hash)
sentence_hash.add(sentence_num);
					if(freq_calc) for(k=0; k<int_sentence_len;
k++) if(vocabulary_count[int_sentence[k]]<65535)
vocabulary_count[int_sentence[k]]++;
					int_sentence[int_sentence_len++]=0;
					int_sentence+=int_sentence_len;
				}
			}
```

```
                int_sentence_len=0;
            }
            else
            {
                if(page[i]==' '||page[i]=='\r') continue;
                word=page+i;
                while(((unsigned char)page[i]>=192&&(unsigned
char)page[i]<=255)||page[i]=='ë'||page[i]=='Ë'||page[i]=='-') i++;
                if(page[i]!=' '&&page[i]!='\r')
                {
                    while(page[i]!=' '&&page[i]!='\r') i++;
                    continue;
                }
                else
                {
                    page[i]='\0';
                    k=(page+i)-word;
                    for(j=0; j<k; j++) word[j]=tolower((unsigned
char)word[j]);
                    if(*word=='-'||word[j-1]=='-') continue;
                    if(j=word_hash.search(word))
int_sentence[int_sentence_len++]=j;
                    else
                    {
                        if(vocabulary_len+1>=vocabulary_maxlen)
                        {
                            unsigned int len=vocabulary_maxlen;
                            endlessmass(vocabulary,
vocabulary_len+1, vocabulary_ext, len);
                            endlessmass(vocabulary_count,
vocabulary_len+1, vocabulary_ext, vocabulary_maxlen);
                        }
                        word_hash.add(++vocabulary_len);
                        vocabulary[vocabulary_len]=new char[k+1];
                        strcpy(vocabulary[vocabulary_len], word);
                        vocabulary_count[vocabulary_len]=0;

    int_sentence[int_sentence_len++]=vocabulary_len;
                    }
                }
            }
        }
        i=(int_sentence-int_page)<<2;
        if(i) outmass(opath, (char*)int_page, i);
        delete [] page;
        delete [] int_page;
}
coding::~coding()
{
```

```
	if(vocabulary==NULL) return;
	unsigned int i,
		j,
		k,
		q;
	char *vf,
		buf[20];
	wchar_t buf_path[MAX_PATH];
	word_hash.save();
	word_hash.close();
	if(use_sentence_hash)
	{
		sentence_hash.save();
		sentence_hash.close();
	}
	wcscpy(buf_path, vocabulary_path);
	wcscat(buf_path, L"vocabulary.txt");
	j=24+(vocabulary_len<<3);
	for(i=1; i<=vocabulary_len; i++) j+=strlen(vocabulary[i]);
	vf=new char[j+1];
	j=0;
	toa(vocabulary_maxlen, buf);
	k=strlen(buf);
	for(i=0; i<k; i++, j++) vf[j]=buf[i];
	vf[j++]='\t';
	toa(sentence_num, buf);
	k=strlen(buf);
	for(i=0; i<k; i++, j++) vf[j]=buf[i];
	vf[j++]='\r';
	vf[j++]='\n';
	for(q=1; q<=vocabulary_len; q++)
	{
		toa(vocabulary_count[q], buf);
		k=strlen(buf);
		for(i=0; i<k; i++, j++) vf[j]=buf[i];
		vf[j++]='\t';
		k=strlen(vocabulary[q]);
		for(i=0; i<k; i++, j++) vf[j]=vocabulary[q][i];
		vf[j++]='\r';
		vf[j++]='\n';
		delete [] vocabulary[q];
	}
	outmass(buf_path, vf, j);
	delete [] vf;
	delete [] vocabulary;
	delete [] vocabulary_count;
	vocabulary=NULL;
}
```

```
//Частотный словарь
class frequency_vocabulary{
	unsigned short int *word_freq,
		**cashe;
	unsigned int *long_list,
		*short_list,
		*cashe_long_list,
		*cashe_short_list,
		long_len,
		short_len,
		cashe_freq,
		cashe_len,
		dpl;
	char **word;
	wchar_t path[MAX_PATH],
		data_path[MAX_PATH],
		*data_path_dpl;
public:
	frequency_vocabulary() {word_freq=NULL;}
	frequency_vocabulary(wchar_t *p, unsigned int f, unsigned int m)
	{
		initialize(p, f, m);
	}
	void initialize(wchar_t *p, unsigned int f, unsigned int m);

	void calculate_doc(wchar_t *p);
	char *get_word(unsigned int w) {return word[w];}
	double get_word_freq(unsigned int w);
	double get_word_unic(unsigned int w);
	double get_word_belong(unsigned int w, double **wr, unsigned int
wrl);

	~frequency_vocabulary();
};
void frequency_vocabulary::initialize(wchar_t *p, unsigned int f,
unsigned int m)
{
	unsigned short int *s;
	unsigned int i,
		j,
		k,
		vf_len;
	char *vf,
		*buf;
	wchar_t buf_path[MAX_PATH];
	wcscpy(path, p);
	wcscpy(data_path, p);
	wcscat(data_path, L"vocabulary_frequency\\");
	dpl=wcslen(data_path);
```

```
	data_path_dpl=data_path+dpl;
	wcscpy(buf_path, p);
	wcscat(buf_path, L"vocabulary.txt");
	if(_waccess(buf_path, 0))
	{
		word_freq=NULL;
		return;
	}
	inmass(buf_path, vf, vf_len);
	if(!vf_len)
	{
		word_freq=NULL;
		return;
	}
	for(i=0, long_len=0; i<vf_len; i++) if(vf[i]=='\n') long_len++;
	word=new char*[long_len];
	word_freq=new unsigned short int[long_len];
	long_list=new unsigned int[long_len];
	for(i=0; vf[i]!='\n'; i++);
	i++;
	for(j=1; i<vf_len; i++, j++)
	{
		buf=vf+i;
		while(vf[i]!='\t') i++;
		vf[i]='\0';
		ato(buf, k);
		word_freq[j]=(unsigned short int)k;
		i++;
		buf=vf+i;
		while(vf[i]!='\r') i++;
		vf[i]='\0';
		k=vf+i-buf+1;
		word[j]=new char[k];
		strcpy(word[j], buf);
		i++;
	}
	delete [] vf;
	for(i=1, short_len=0; i<long_len; i++)
	{
		if(word_freq[i]>=f) long_list[i]=++short_len;
		else long_list[i]=0;
	}
	short_list=new unsigned int[short_len+1];
	for(i=1; i<long_len; i++) if(long_list[i])
short_list[long_list[i]]=i;
	if(m)
	{
		s=new unsigned short int[long_len];
		*s=0;
```

```
            for(i=1; i<long_len; i++) s[i]=word_freq[i];
            qsort(s, long_len, sizeof(unsigned short int),
short_compare);
            j=m/((short_len+2)<<1);
            for(i=j-1; i>0; i--) if(s[j]!=s[i]) break;
            cashe_len=i+1;
            cashe_freq=i?s[i]:0;
            delete [] s;
            cashe_long_list=new unsigned int[short_len+1];
            for(i=1, j=0; i<=short_len; i++)
            {
                  if(word_freq[short_list[i]]>=cashe_freq)
cashe_long_list[i]=++j;
                  else cashe_long_list[i]=0;
            }
            cashe_short_list=new unsigned int[cashe_len+1];
            for(i=1; i<=short_len; i++) if(cashe_long_list[i])
cashe_short_list[cashe_long_list[i]]=i;
      }
      else
      {
            cashe_freq=0;
            cashe_len=0;
      }
      wcscat(data_path, L"1");
      buf=new char[20];
      if(_waccess(data_path, 0))
      {
            *data_path_dpl=L'\0';
            crtdir(data_path);
            unsigned short int *cashe_buf=new unsigned short
int[short_len];
            for(i=0; i<short_len; i++) cashe_buf[i]=0;
            for(i=1; i<=short_len; i++)
            {
                  toa(i, buf);
                  k=strlen(buf)+1;
                  MultiByteToWideChar(CP_ACP, NULL, buf, k,
data_path_dpl, k);
                  ofstream out(data_path, ios::binary);
                  if(!out) throw data_path;
                  out.write((char*)cashe_buf, short_len<<1);
                  out.close();
            }
            delete [] cashe_buf;
            if(cashe_len)
            {
                  cashe=new unsigned short int*[cashe_len+1];
                  for(i=1; i<=cashe_len; i++)
```

```
                {
                        cashe[i]=new unsigned short int[short_len];
                        for(j=0; j<short_len; j++) cashe[i][j]=0;
                }
        }
    }
    else
    {
        if(cashe_len)
        {
                cashe=new unsigned short int*[cashe_len+1];
                for(i=1; i<=cashe_len; i++)
                {
                        cashe[i]=new unsigned short int[short_len];
                        toa(cashe_short_list[i], buf);
                        k=strlen(buf)+1;
                        MultiByteToWideChar(CP_ACP, NULL, buf, k,
data_path_dpl, k);
                        ifstream in(data_path, ios::binary);
                        if(!in) throw data_path;
                        in.read((char*)cashe[i], short_len<<1);
                        in.close();
                }
        }
    }
    *data_path_dpl=L'\0';
    delete [] buf;
}
void frequency_vocabulary::calculate_doc(wchar_t *p)
{
    unsigned short int val;
    unsigned int doc_len,
        *doc,
        i,
        j,
        k,
        q,
        l;
    char buf[20];
    if(_waccess(p, 0)) return;
    ifstream in(p, ios::binary);
    if(!in) throw p;
    in.seekg(0, ios::end);
    doc_len=(unsigned int)in.tellg()/sizeof(*doc);
    doc=new unsigned int[doc_len];
    in.seekg(0, ios::beg);
    in.read((char*)doc, doc_len*sizeof(*doc));
    in.close();
    for(i=0; i<doc_len; i++)
```

```
	{
		for(j=0; doc[i]!=0; i++)
		{
			if(long_list[doc[i]])
			{
				if(j)
				{
					if(doc[j-1]!=doc[i]) doc[j++]=doc[i];
				}
				else doc[j++]=doc[i];
			}
		}
		if(j>1)
		{
			for(k=0; k<j; k++) for(q=0; q<j; q++)
			{
				if(word_freq[doc[k]]>=cashe_freq)
				{

	if(cashe[cashe_long_list[long_list[doc[k]]]][long_list[doc[q]]-
1]<65535) cashe[cashe_long_list[long_list[doc[k]]]][long_list[doc[q]]-
1]++;
				}
				else
				{
					toa(long_list[doc[k]], buf);
					l=strlen(buf)+1;
					MultiByteToWideChar(CP_ACP, NULL, buf, l,
data_path_dpl, l);
					fstream note(data_path,
ios::in|ios::out|ios::ate|ios::binary);
					if(!note) throw data_path;
					note.seekg((long_list[doc[q]]-1)<<1,
ios::beg);
					note.read((char*)&val, sizeof(val));
					if(val<65535)
					{
						val++;
						note.seekp((long_list[doc[q]]-1)<<1,
ios::beg);
						note.write((char*)&val, sizeof(val));
					}
					note.close();
				}
			}
		}
		*data_path_dpl=L'\0';
	}
	delete [] doc;
```

```
}
double frequency_vocabulary::get_word_freq(unsigned int w)
{
	unsigned int i;
	unsigned short int val;
	char buf[20];
	wchar_t buf_path[MAX_PATH];
	if(long_list[w])
	{
		wcscpy(buf_path, data_path);
		toa(long_list[w], buf);
		i=strlen(buf)+1;
		MultiByteToWideChar(CP_ACP, NULL, buf, i, buf_path+dpl, i);
		ifstream in(buf_path, ios::binary);
		if(!in) throw buf_path;
		in.seekg(((long_list[w]-1)<<1), ios::beg);
		in.read((char*)&val, sizeof(val));
		in.close();
		if(val>100) return 1.0;
		else return (double)val/100.0;
	}
	else return 0.0;
}
double frequency_vocabulary::get_word_unic(unsigned int w)
{
	unsigned int i,
		val1=0,
		val2=0;
	unsigned short int *mass;
	char buf[20];
	wchar_t buf_path[MAX_PATH];
	if(long_list[w])
	{
		wcscpy(buf_path, data_path);
		toa(long_list[w], buf);
		i=strlen(buf)+1;
		MultiByteToWideChar(CP_ACP, NULL, buf, i, buf_path+dpl, i);
		ifstream in(buf_path, ios::binary);
		if(!in) throw buf_path;
		mass=new unsigned short int[short_len];
		in.read((char*)mass, short_len<<1);
		in.close();
		for(i=0; i<short_len; i++)
		{
			if(word_freq[short_list[i+1]]>=25)
			{
				if(mass[i])
				{
					val1++;
```

```
                              val2++;
                        }
                        else val2++;
                  }
            }
            delete [] mass;
            return 1.0-(double)val1/(double)val2;
      }
      else return 1.0;
}
double frequency_vocabulary::get_word_belong(unsigned int w, double
**wr, unsigned int wrl)
{
      unsigned int i,
            val1=0,
            val2=0;
      unsigned short int *mass;
      char buf[20];
      wchar_t buf_path[MAX_PATH];
      if(long_list[w])
      {
            wcscpy(buf_path, data_path);
            toa(long_list[w], buf);
            i=strlen(buf)+1;
            MultiByteToWideChar(CP_ACP, NULL, buf, i, buf_path+dpl, i);
            ifstream in(buf_path, ios::binary);
            if(!in) throw buf_path;
            mass=new unsigned short int[short_len];
            in.read((char*)mass, short_len<<1);
            in.close();
            for(i=0; i<wrl; i++) if(wr[i][3]>0.5&&long_list[(unsigned
int)wr[i][0]]) if(mass[long_list[(unsigned int)wr[i][0]]-1])
            {
                  val1++;
                  val2++;
            }
            else val2++;
            delete [] mass;
            return (double)val1/(double)val2;
      }
      else return 0.0;
}
frequency_vocabulary::~frequency_vocabulary()
{
      unsigned int i,
            j;
      char buf[20];
      if(word_freq==NULL) return;
      if(cashe_len)
```

```
	{
		for(i=1; i<=cashe_len; i++)
		{
			toa(cashe_short_list[i], buf);
			j=strlen(buf)+1;
			MultiByteToWideChar(CP_ACP, NULL, buf, j,
data_path_dpl, j);
			ofstream out(data_path, ios::binary);
			if(!out) throw data_path;
			out.write((char*)cashe[i], short_len<<1);
			out.close();
			delete [] cashe[i];
		}
		delete [] cashe;
		delete [] cashe_long_list;
		delete [] cashe_short_list;
	}
	delete [] word_freq;
	delete [] long_list;
	delete [] short_list;
	for(i=1; i<long_len; i++) delete [] word[i];
	delete [] word;
	word_freq=NULL;
}

//Коллекции документов
class collection{
	unsigned int collections_len, //Длинна файла коллекций.
		themes_list_len, //Длина списка тем, по которым мы
собираемся добавлять документы.
		collections_num, //Количество коллекций.
		themes_num, //Количество тем в данной коллекции.
		docs_num, //Количество документов в данной коллекции.
		max_docs_theme, //Максимальное количество документов на тему
в данной коллекции.
		links_num, //Количество просканированных ссылок при сборе
данной коллекции.
		sentence_num, //Количество просканированных предложений при
сборе данной коллекции.
		themes_list_num, //Количество тем в списке, по которым мы
собираемся добавлять документы.
		words; //Колличество слов в данной колекции.
	char name[41], //Имя коллекции.
		*collections, //Указатель на массив, в который загружается
файл с данными о коллекциях.
		*themes_list; //Список тем по которым мы собираемся
добавлять документы.
	wchar_t path[59], //Путь к коллекции.
		doc_path[64], //Путь к документам коллекции.
```

```
            theme_path[75]; //Путь к теме, с которой мы работаем в
данный момент.
      my_hash_str theme_hash,
            links_hash,
            server_except_hash;
      scaning s1;
      lemma l1;
      coding code;
      frequency_vocabulary f1;

      static const unsigned int collections_start=89; //Место откуда
начинаются записи о коллекциях.

      void affix_settings();
      void cl_hash(my_hash_str &hash, wchar_t *p, unsigned int s,
unsigned int d);
      void prepare_list(char *notes_list, unsigned int &notes_list_len,
unsigned int &notes_list_num);
      void load_server_except_hash();
      void load_hash_tables(unsigned int t, unsigned int dt);
      void add_theme(char *theme, unsigned int dt);

public:
      //Настройки коллекции.
      settings col_settings;

      setting<unsigned int> use_theme_hash,
            theme_row_size,
            yandex_xml,
            exact_query,
            save_links,
            use_server_except,
            use_links_hash,
            links_row_size,
            delete_index,
            delete_page,
            use_sentence_scan,
            min_length,
            min_words,
            save_open_tag,
            min_sentences,
            delete_scan_page,
            use_lemmatization,
            delete_lemma_page,
            use_word_coding,
            word_row_size,
            use_sentence_sort,
            use_sentence_hash,
            sentence_row_size,
```

```
            max_all_hash_size,
            missing_symbol,
            min_frequency,
            freq_vocabulary_mem;
      setting<char *> yandex_xml_options;
      collection();

      unsigned int get_collections_num() { return collections_num; }
      unsigned int get_themes_num() { return themes_num; }
      unsigned int get_docs_num() { return docs_num; }
      unsigned int get_max_docs_theme() { return max_docs_theme; }
      unsigned int get_links_num() { return links_num; }
      unsigned int get_sentence_num() { return sentence_num; }
      unsigned int get_themes_list_num() { return themes_list_num; }
      unsigned int get_words() { return words; }
      wchar_t *get_path() { return path; }

      bool isexist();
      void print();
      void create(char *n);
      void load(unsigned int c);
      void add_list(unsigned int fr, unsigned int to, unsigned int dt);
      void make_stages(unsigned int st);
      void prepare_themes_list();
      unsigned int gt_frequency(unsigned int n);
      void freq_calculate();
      void remove_stages(unsigned int st);
      void remove();

      ~collection();
};
collection::collection()
{
      name[0]='\0';
      wcscpy(path, L"data");
      crtdir(path);
      wcscat(path, L"\\collections");
      crtdir(path);
      wcscat(path, L"\\");
      inmass(L"data\\collections.txt", collections, collections_len,
300);
      if(collections_len==0)
      {
            strcpy(collections, "Имя | Кол-во тем | Кол-во документов |
Макс. док. на тему | Ссылок | Предложений | слов\r\n");
            collections_len=collections_start;
      }
      collections_num=0;
```

```
      for(unsigned int i=collections_start; i<collections_len; i++)
if(collections[i]=='\n') collections_num++;
}
void collection::affix_settings()
{
      wchar_t settings_path[74];

      use_theme_hash.initialize("use_theme_hash", 0, 1, 1);
      theme_row_size.initialize("theme_row_size", 0, 4294967295, 0);
      yandex_xml.initialize("yandex_xml", 0, 1, 0);
      yandex_xml_options.initialize("yandex_xml_options", "\0", 220,
"\0");
      exact_query.initialize("exact_query", 0, 1, 1);
      delete_index.initialize("delete_index", 0, 1, 0);
      delete_page.initialize("delete_page", 0, 1, 0);
      use_sentence_scan.initialize("use_sentence_scan", 0, 1, 1);
      min_length.initialize("min_length", 1, 4294967295, 8);
      min_words.initialize("min_words", 1, 4294967295, 2);
      save_open_tag.initialize("save_open_tag", 0, 1, 1);
      min_sentences.initialize("min_sentences", 1, 4294967295, 10);
      use_lemmatization.initialize("use_lemmatization", 0, 1, 1);
      delete_scan_page.initialize("delete_scan_page", 0, 1, 0);
      delete_lemma_page.initialize("delete_lemma_page", 0, 1, 0);
      use_word_coding.initialize("use_word_coding", 0, 1, 1);
      use_sentence_sort.initialize("use_sentence_sort", 0, 1, 1);
      use_server_except.initialize("use_server_except", 0, 1, 1);
      use_links_hash.initialize("use_links_hash", 0, 1, 1);
      save_links.initialize("save_links", 0, 1, 1);
      use_sentence_hash.initialize("use_sentence_hash", 0, 1, 1);
      max_all_hash_size.initialize("max_all_hash_size", 0, 4294967295,
1072166814);
      links_row_size.initialize("links_row_size", 0, 4294967295, 0);
      sentence_row_size.initialize("sentence_row_size", 0, 4294967295,
0);
      word_row_size.initialize("word_row_size", 0, 4294967295, 0);
      missing_symbol.initialize("missing_symbol", 0, 255, 152);
      min_frequency.initialize("min_frequency", 0, 65535, 2);
      freq_vocabulary_mem.initialize("freq_vocabulary_mem", 0,
4294967295, 1073741824);

      wcscpy(settings_path, path);
      wcscat(settings_path, L"settings.txt");

      col_settings.initialize(settings_path, 30);
      col_settings.add(yandex_xml);
      col_settings.add(yandex_xml_options);
      col_settings.add(exact_query);
      col_settings.add(delete_index);
      col_settings.add(use_sentence_scan);
```

```
	col_settings.add(min_length);
	col_settings.add(min_words);
	col_settings.add(save_open_tag);
	col_settings.add(min_sentences);
	col_settings.add(use_lemmatization);
	col_settings.add(delete_page);
	col_settings.add(delete_scan_page);
	col_settings.add(delete_lemma_page);
	col_settings.add(use_word_coding);
	col_settings.add(use_sentence_sort);
	col_settings.add(use_theme_hash);
	col_settings.add(use_server_except);
	col_settings.add(use_links_hash);
	col_settings.add(save_links);
	col_settings.add(use_sentence_hash);
	col_settings.add(max_all_hash_size);
	col_settings.add(theme_row_size);
	col_settings.add(links_row_size);
	col_settings.add(sentence_row_size);
	col_settings.add(word_row_size);
	col_settings.add(missing_symbol);
	col_settings.add(min_frequency);
	col_settings.add(freq_vocabulary_mem);
}
void collection::prepare_list(char *notes_list, unsigned int
&notes_list_len, unsigned int &notes_list_num)
{
	unsigned int i=0,
		j=0;
	notes_list_num=0;
	bool space=0,
		note=0;
	for(; i<notes_list_len; i++)
	{
		if(notes_list[i]=='\n'&&note)
		{
			notes_list[j++]='\r';
			notes_list[j++]='\n';
			note=space=0;
			notes_list_num++;
		}
		else if(notes_list[i]=='
'||notes_list[i]=='\t'||notes_list[i]=='\r'||notes_list[i]=='\n')
		{
			if(note) space=1;
		}
		else if(notes_list[i]=='/'&&notes_list[i+1]=='/')
		{
			i+=2;
```

```
                while(notes_list[i]!='\r'||notes_list[i]!='\0') i++;
            }
            else
            {
                note=1;
                if(space)
                {
                    notes_list[j++]=' ';
                    space=0;
                }
                notes_list[j++]=notes_list[i];
            }
        }
        if(notes_list[j-1]!='\n'&&note)
        {
            notes_list_num++;
            notes_list[j++]='\r';
            notes_list[j++]='\n';
        }
        notes_list_len=j;
}
void collection::load_server_except_hash()
{
        unsigned int server_except_list_len,
            server_except_list_num,
            server_count=1,
            i,
            hash_len=1;
        char *server_except_list,
            *server,
            command[100]="rmdir /s/q data\\collections\\";
        wchar_t server_except_path[80];
        wcscpy(server_except_path, path);
        wcscat(server_except_path, L"server_except_list.txt");
        inmass(server_except_path, server_except_list,
server_except_list_len, 2);
        server_except_list[server_except_list_len]='\0';
        prepare_list(server_except_list, server_except_list_len,
server_except_list_num);
        wcscpy(server_except_path, path);
        wcscat(server_except_path, L"server_except_hash\\");
        if(!_waccess(server_except_path, 0))
        {
            strcat(command, name);
            strcat(command, "\\server_except_hash");
            system(command);
        }
        while(hash_len<server_except_list_num+2) hash_len<<=1;
        hash_len--;
```

```
	server_except_hash.rebuild(server_except_path, hash_len);
	for(i=0; server_count<=server_except_list_num; server_count++)
	{
		server=server_except_list+i;
		for(; server_except_list[i]!='\r'; i++)
server_except_list[i]=tolower(server_except_list[i]);
		server_except_list[i]='\0';
		i+=2;
		if(!server_except_hash.search(server))
server_except_hash.add(server_count);
	}
	delete [] server_except_list;
}
void collection::add_theme(char *theme, unsigned int dt)
{
	unsigned int theme_num=0,
		i,
		j,
		doc_theme;
	char buf[30],
		*server,
		*serverreq,
		command[300];
	wchar_t wbuf[20],
		buf_path[MAX_PATH],
		wcommand[300];
	fstream prop;
	if(use_theme_hash.get())
	{
		theme_num=theme_hash.search(theme);
		if(theme_num==0)
		{
			theme_num=themes_num+1;
			theme_hash.add(theme_num);
		}
	}
	if(theme_num==0) theme_num=themes_num+1;
	toa(theme_num, buf);
	i=strlen(buf)+1;
	MultiByteToWideChar(CP_ACP, NULL, buf, i, wbuf, i);
	wcscpy(theme_path, doc_path);
	wcscat(theme_path, wbuf);
	wcscat(theme_path, L"\\");
	wcscpy(buf_path, theme_path);
	wcscat(buf_path, L"properties.txt");
	if(theme_num<=themes_num)
	{
		prop.open(buf_path, ios::in|ios::out|ios::ate|ios::binary);
		if(!prop) throw buf_path;
```

```
            prop.seekg(0, ios::beg);
            prop.read((char *)&doc_theme, sizeof(doc_theme));
      }
      else
      {
            themes_num++;
            doc_theme=0;
            crtdir(theme_path);
            prop.open(buf_path, ios::out|ios::ate|ios::binary);
            if(!prop) throw buf_path;
            prop.write((char *)&doc_theme, sizeof(doc_theme));
            prop.put('\t');
            prop.write(theme, strlen(theme));
      }
      if(doc_theme<dt)
      {
            page p1(theme, theme_path, exact_query.get(),
save_links.get(), yandex_xml.get(), yandex_xml_options.get());
            while(doc_theme<dt)
            {
                  cout << (serverreq=p1.get_serverreq()) << endl;
                  if(*serverreq=='\0') break;
                  if(use_server_except.get())
                  {
                        j=strlen(serverreq);
                        server=serverreq;
                        for(i=0; serverreq[i]!='\0'&&serverreq[i]!='/';
i++) ;
                        serverreq[i]='\0';
                        if(server_except_hash.search(server))
                        {
                              cout << "Сервер " << server << " в списке
исключений" << endl;
                              continue;
                        }
                        if(j>i) serverreq[i]='/';
                  }
                  if(use_links_hash.get())
if(links_hash.search(serverreq))
                  {
                        cout << "Документ по ссылке " << serverreq << "
уже находится в коллекции" << endl;
                        continue;
                  }
                  wcscpy(buf_path, p1.get_new_page(doc_theme+1));
                  if(*buf_path!='\0')
                  {
                        if(use_links_hash.get())
links_hash.add(++links_num);
```

```
                        if(use_sentence_scan.get())
                        {
                              if(s1.scan(buf_path,
theme)<min_sentences.get())
                              {
                                    cout << "В документе найдено меньше
предложений чем необходимо" << endl;
                                    _wremove(buf_path);
                                    buf_path[wcslen(buf_path)-4]=L'\0';
                                    wcscat(buf_path, L"txt");
                                    _wremove(buf_path);
                                    continue;
                              }
                              if(delete_page.get()) _wremove(buf_path);
                              if(use_lemmatization.get())
                              {
                                    buf_path[wcslen(buf_path)-4]=L'\0';
                                    wcscat(buf_path, L"txt");
                                    l1.lem(buf_path);
                                    if(delete_scan_page.get())
_wremove(buf_path);
                                    if(use_word_coding.get())
                                    {
                                          buf_path[wcslen(buf_path)-
4]=L'\0';

                                          wcscat(buf_path, L"_lem.txt");
                                          code.code_page(buf_path);
                                          if(delete_lemma_page.get())
_wremove(buf_path);
                                    }
                              }
                        }
                  }
                  else continue;
                  doc_theme++;
                  docs_num++;
            }
      }
      if(delete_index.get())
      {
            wcscpy(wcommand, L"rmdir /s/q ");
            wcscat(wcommand, theme_path);
            wcscat(wcommand, L"index");
            i=wcslen(wcommand)+1;
            WideCharToMultiByte(CP_ACP, NULL, wcommand, i, command, i,
NULL, NULL);
            system(command);
      }
      prop.seekp(0, ios::beg);
```

```
	prop.write((char *)&doc_theme, sizeof(doc_theme));
	prop.close();
}
bool collection::isexist()
{
	if(name[0]=='\0') return false;
	else return true;
}
void collection::print()
{
	unsigned int num=0;
	for(unsigned int i=collections_start+1; i<collections_len;)
	{
		cout << ++num << ". ";
		while(collections[i]!='\"') cout << collections[i++];
		cout << endl;
		while(collections[i]!='\n') i++;
		i+=2;
	}
}
void collection::create(char *n)
{
	unsigned int i,
		nl=strlen(n);
	char fname[43]="\"";
	wchar_t themes_list_path[74],
		except_servers_list_path[80],
		wname[41];
	if(strlen(n)>40||strlen(n)<1) throw n;
	strcat(fname, n);
	strcat(fname, "\"");
	low_finding identical(fname, collections, collections_len);
	if(identical.find()!=-1) throw n;
	for(i=0; i<nl; i++)
if(!((n[i]>='A'&&n[i]<='Z')||(n[i]>='a'&&n[i]<='z')||
	(n[i]>='0'&&n[i]<='9')||n[i]=='-'||n[i]=='_'||n[i]=='
'||(n[i]>='A'&&n[i]<='я')||
	n[i]=='ё'||n[i]=='Ё')) break;
	if(i<strlen(n)) throw n;
	strcpy(name, n);
	words=themes_list_num=themes_num=docs_num=max_docs_theme=links_nu
m=sentence_num=0;
	MultiByteToWideChar(CP_ACP, NULL, name, nl, wname, nl);
	wname[nl]='\0';
	wcscat(path, wname);
	crtdir(path);
	wcscat(path, L"\\");
	wcscpy(doc_path, path);
	wcscat(doc_path, L"docs");
```

```
	crtdir(doc_path);
	wcscat(doc_path, L"\\");
	wcscpy(themes_list_path, path);
	wcscat(themes_list_path, L"themes_list.txt");
	ofstream themes_list_file(themes_list_path, ios::binary);
	if(!themes_list_file) throw themes_list_path;
	themes_list_file.write("//Напишите сюда темы, документы по
которым хотите добавить в коллекцию.\r\n//Всё что написано на одной
строке считается темой, кроме комментариев начинающихся с // и
действующих до конца строки.\r\n//Лишние пробельные символы
игнорируются, так что их можно использовать для
структурирования.\r\n//Длинна темы не более 400 символов из за
ограничения яндекса.\r\n\r\n", 355);
	themes_list_file.close();
	wcscpy(except_servers_list_path, path);
	wcscat(except_servers_list_path, L"server_except_list.txt");
	ofstream except_servers_list_file(except_servers_list_path,
ios::binary);
	if(!except_servers_list_file) throw except_servers_list_path;
	except_servers_list_file.write("//Напишите сюда сервера, которые
нужно игнорировать (Не указывайте протокол \"http://\").\r\n//Всё что
написано на одной строке считается сервером, кроме комментариев
начинающихся с // и действующих до конца строки.\r\n//Лишние
пробельные символы игнорируются, так что их можно использовать для
структурирования.\r\n\r\n", 310);
	except_servers_list_file.close();
	themes_list=NULL;
	affix_settings();
	col_settings.def();
	col_settings.save();
	my_hash_str::set_max_size(max_all_hash_size.get());
}
void collection::load(unsigned int c)
{
	if(!(c>0&&c<=collections_num))
	{
		throw "Коллекции с таким номером не существует.";
	}
	unsigned int pos=collections_start,
		erase,
		i,
		nl;
	char buf[11];
	wchar_t wname[41];
	if(c!=1)
	{
		for(i=1; i<c; pos++) if(collections[pos]=='\n') i++;
		pos++;
	}
```

```
      erase=pos+1;
      for(i=0; collections[erase]!='\"'; erase++, i++)
name[i]=collections[erase];
      name[i]='\0';
      cout << "\t" << name << endl << endl;
      erase+=2;
      for(i=0; collections[erase]!=' '; erase++, i++)
buf[i]=collections[erase];
      buf[i]='\0';
      ato(buf, themes_num);
      cout << "\tТемы: " << buf << endl;
      erase++;
      for(i=0; collections[erase]!=' '; erase++, i++)
buf[i]=collections[erase];
      buf[i]='\0';
      ato(buf, docs_num);
      cout << "\tДокументы: " << buf << endl;
      erase++;
      for(i=0; collections[erase]!=' '; erase++, i++)
buf[i]=collections[erase];
      buf[i]='\0';
      ato(buf, max_docs_theme);
      cout << "\tМакс. док. на тему: " << buf << endl;
      erase++;
      for(i=0; collections[erase]!=' '; erase++, i++)
buf[i]=collections[erase];
      buf[i]='\0';
      ato(buf, links_num);
      cout << "\tСсылки: " << buf << endl;
      erase++;
      for(i=0; collections[erase]!=' '; erase++, i++)
buf[i]=collections[erase];
      buf[i]='\0';
      ato(buf, sentence_num);
      cout << "\tПредложения: " << sentence_num << endl;
      erase++;
      for(i=0; collections[erase]!='\r'; erase++, i++)
buf[i]=collections[erase];
      buf[i]='\0';
      ato(buf, words);
      cout << "\tСлова: " << words << endl << endl;
      erase+=2;
      nl=strlen(name)+1;
      MultiByteToWideChar(CP_ACP, NULL, name, nl, wname, nl);
      wcscat(path, wname);
      wcscat(path, L"\\");
      wcscpy(doc_path, path);
      wcscat(doc_path, L"docs\\");
```

```
	while(erase<collections_len)
collections[pos++]=collections[erase++];
	collections_len=pos;
	affix_settings();
	col_settings.load();
	themes_list_num=0;
	themes_list=NULL;
	my_hash_str::set_max_size(max_all_hash_size.get());
}
void collection::prepare_themes_list()
{
	wchar_t themes_list_path[74];
	wcscpy(themes_list_path, path);
	wcscat(themes_list_path, L"themes_list.txt");
	inmass(themes_list_path, themes_list, themes_list_len, 2);
	themes_list[themes_list_len]='\0';
	prepare_list(themes_list, themes_list_len, themes_list_num);
}
void collection::add_list(unsigned int fr, unsigned int to, unsigned
int dt)
{
	unsigned int theme_count=1,
		i,
		j,
		themes_list_pos=0;
	char *theme;
	wchar_t buf_path[MAX_PATH];
	if(themes_list_num==0) throw "Список тем не подготовлен, или в
нём нет ни одной темы.";
	if(dt==0) return;
	if(!(fr>0&&fr<=to&&to<=themes_list_num)) return;
	if(max_docs_theme<dt) max_docs_theme=dt;
	if(use_theme_hash.get())
	{
		wcscpy(buf_path, path);
		wcscat(buf_path, L"theme_hash\\");
		if(!(i=theme_row_size.get()))
		{
			i=1;
			j=themes_num+to-fr+2;
			while(i<j) i<<=1;
			i--;
		}
		if(theme_hash.is_exist(buf_path)==i)
theme_hash.load(buf_path);
		else theme_hash.rebuild(buf_path, i);
	}
	if(use_links_hash.get())
	{
```

```
            wcscpy(buf_path, path);
            wcscat(buf_path, L"links_hash\\");
            if(!(i=links_row_size.get()))
            {
                  i=1;
                  j=links_num+(to-fr+1)*dt+1;
                  while(i<j) i<<=1;
                  i--;
            }
            if(links_hash.is_exist(buf_path)==i)
links_hash.load(buf_path);
            else links_hash.rebuild(buf_path, i);
      }
      if(use_server_except.get()) load_server_except_hash();
      if(use_sentence_scan.get()) s1.initialize(min_length.get(),
min_words.get(), (bool)save_open_tag.get(), (unsigned
char)missing_symbol.get());
      if(use_lemmatization.get()) l1.initialize();
      if(use_word_coding.get()) code.initialize(path,
word_row_size.get(), words+100000, use_sentence_sort.get(),
sentence_row_size.get(), sentence_num+(to-fr+1)*100,
use_sentence_hash.get());
      while(theme_count<fr) if(themes_list[themes_list_pos++]=='\n')
theme_count++;
      for(; theme_count<=to; theme_count++)
      {
            for(i=themes_list_pos; themes_list[i]!='\r'; ) i++;
            j=i-themes_list_pos+1;
            if(j>400) continue;
            theme=new char[j];
            for(j=0; themes_list_pos<i; j++, themes_list_pos++)
theme[j]=themes_list[themes_list_pos];
            theme[j]='\0';
            themes_list_pos+=2;
            add_theme(theme, dt);
            delete [] theme;
      }
      if(use_sentence_scan.get()) s1.~scaning();
      if(use_word_coding.get())
      {
            words=code.get_words_num();
            sentence_num=code.get_sentence_num();
            code.~coding();
      }
      if(use_theme_hash.get())
      {
            theme_hash.save();
            theme_hash.close();
      }
```

```
	if(use_links_hash.get())
	{
		links_hash.save();
		links_hash.close();
	}
	if(use_server_except.get()) server_except_hash.close();
	if(use_lemmatization.get()) l1.~lemma();
	delete [] themes_list;
	themes_list=NULL;
	themes_list_num=themes_list_len=0;
}
void collection::make_stages(unsigned int st)
{
	unsigned int theme_num,
		doc_theme,
		bs,
		bl,
		bk,
		i;
	char buf[20],
		theme[401];
	wchar_t buf_path[MAX_PATH],
		*buf_path_bs;
	if(st&1) s1.initialize(min_length.get(), min_words.get(),
(bool)save_open_tag.get(), (unsigned char)missing_symbol.get());
	if(st&2) l1.initialize();
	if(st&4) code.initialize(path, word_row_size.get(), words+100000,
use_sentence_sort.get(), sentence_row_size.get(), docs_num*100,
use_sentence_hash.get());
	wcscpy(buf_path, doc_path);
	bs=wcslen(buf_path);
	buf_path_bs=buf_path+bs;
	for(theme_num=1; theme_num<=themes_num; theme_num++)
	{
		toa(theme_num, buf);
		bl=strlen(buf)+1;
		MultiByteToWideChar(CP_ACP, NULL, buf, bl, buf_path_bs, bl);
		wcscat(buf_path, L"\\");
		wcscat(buf_path, L"properties.txt");
		if(!_waccess(buf_path, 0))
		{
			ifstream prop(buf_path, ios::binary);
			if(!prop) throw buf_path;
			prop.read((char *)&doc_theme, sizeof(doc_theme));
			prop.seekg(1, ios::cur);
			prop.read(theme, 401);
			theme[prop.gcount()]='\0';
			prop.close();
		}
```

```
            else continue;
            for(i=1; i<=doc_theme; i++)
            {
                  buf_path[bs+bl]=L'\0';
                  toa(i, buf);
                  bk=strlen(buf);
                  MultiByteToWideChar(CP_ACP, NULL, buf, bk,
buf_path_bs+bl, bk);
                  bk+=bs+bl;
                  buf_path[bk]=L'\0';
                  if(st&1)
                  {
                        wcscat(buf_path, L".html");
                        if(!_waccess(buf_path, 0))
                        {
                              wcout << L"Сканирование " << buf_path <<
endl;
                              s1.scan(buf_path, theme);
                              if(delete_page.get()) _wremove(buf_path);
                        }
                        buf_path[bk]=L'\0';
                  }
                  if(st&2)
                  {
                        wcscat(buf_path, L".txt");
                        if(!_waccess(buf_path, 0))
                        {
                              wcout << L"Лемматизация " << buf_path <<
endl;
                              l1.lem(buf_path);
                              if(delete_scan_page.get())
_wremove(buf_path);
                        }
                        buf_path[bk]=L'\0';
                  }
                  if(st&4)
                  {
                        wcscat(buf_path, L"_lem.txt");
                        if(!_waccess(buf_path, 0))
                        {
                              wcout << L"Кодирование " << buf_path <<
endl;
                              code.code_page(buf_path);
                              if(delete_lemma_page.get())
_wremove(buf_path);
                        }
                  }
            }
      }
```

```
	if(st&1) s1.~scaning();
	if(st&2) l1.~lemma();
	if(st&4)
	{
		words=code.get_words_num();
		sentence_num=code.get_sentence_num();
		code.~coding();
	}
	cout << "\n\n";
}
unsigned int collection::gt_frequency(unsigned int n)
{
	unsigned int mas_len,
		i,
		j,
		x=0;
	char *mas,
		*buf;
	wchar_t p[MAX_PATH];
	wcscpy(p, path);
	wcscat(p, L"vocabulary.txt");
	if(_waccess(p, 0)) return 0;
	inmass(p, mas, mas_len);
	if(!mas_len) return 0;
	for(i=0; mas[i]!='\n'; i++);
	i++;
	for(; i<mas_len; i++)
	{
		buf=mas+i;
		while(mas[i]!='\t') i++;
		mas[i]='\0';
		if(ato(buf, j)>=n) x++;
		while(mas[i]!='\n') i++;
	}
	delete [] mas;
	return x;
}
void collection::freq_calculate()
{
	unsigned int theme_num,
		doc_theme,
		bs,
		bl,
		bk,
		i;
	char buf[20];
	wchar_t buf_path[MAX_PATH],
		*buf_path_bs;
```

```
	f1.initialize(path, min_frequency.get(),
freq_vocabulary_mem.get());
	wcscpy(buf_path, doc_path);
	bs=wcslen(buf_path);
	buf_path_bs=buf_path+bs;
	for(theme_num=1; theme_num<=themes_num; theme_num++)
	{
		toa(theme_num, buf);
		bl=strlen(buf)+1;
		MultiByteToWideChar(CP_ACP, NULL, buf, bl, buf_path_bs, bl);
		wcscat(buf_path, L"\\");
		wcscat(buf_path, L"properties.txt");
		if(!_waccess(buf_path, 0))
		{
			ifstream prop(buf_path, ios::binary);
			if(!prop) throw buf_path;
			prop.read((char *)&doc_theme, sizeof(doc_theme));
			prop.close();
		}
		else continue;
		for(i=1; i<=doc_theme; i++)
		{
			buf_path[bs+bl]=L'\0';
			toa(i, buf);
			bk=strlen(buf);
			MultiByteToWideChar(CP_ACP, NULL, buf, bk,
buf_path_bs+bl, bk);
			bk+=bs+bl;
			buf_path[bk]=L'\0';
			wcscat(buf_path, L"_code.txt");
			if(!_waccess(buf_path, 0))
			{
				wcout << L"Рассчёт частот для " << buf_path <<
endl;
				f1.calculate_doc(buf_path);
			}
			buf_path[bk]=L'\0';
		}
	}
	f1.~frequency_vocabulary();
	cout << "\n\n";
}
void collection::remove_stages(unsigned int st)
{
	unsigned int i,
		j,
		bs,
		bl,
		bk,
```

```
            doc_theme;
      char buf[20],
            command[300]="rmdir /s/q ",
            *command_11=command+11;
      wchar_t buf_path[300],
            *buf_path_bs;
      wcscpy(buf_path, doc_path);
      bs=wcslen(buf_path);
      buf_path_bs=buf_path+bs;
      for(i=1; i<=themes_num; i++)
      {
            toa(i, buf);
            bl=strlen(buf)+1;
            MultiByteToWideChar(CP_ACP, NULL, buf, bl, buf_path_bs, bl);
            wcscat(buf_path, L"\\");
            wcscat(buf_path, L"properties.txt");
            if(!_waccess(buf_path, 0))
            {
                  ifstream in(buf_path, ios::binary);
                  if(!in) throw buf_path;
                  in.read((char*)&doc_theme, 4);
                  in.close();
            }
            else continue;
            for(j=1; j<=doc_theme; j++)
            {
                  toa(j, buf);
                  bk=strlen(buf);
                  MultiByteToWideChar(CP_ACP, NULL, buf, bk,
buf_path_bs+bl, bk);
                  bk+=bs+bl;
                  buf_path[bk]=L'\0';
                  if(st&1)
                  {
                        wcscat(buf_path, L".html");
                        if(!_waccess(buf_path, 0)) _wremove(buf_path);
                        buf_path[bk]=L'\0';
                  }
                  if(st&2)
                  {
                        wcscat(buf_path, L".txt");
                        if(!_waccess(buf_path, 0)) _wremove(buf_path);
                        buf_path[bk]=L'\0';
                  }
                  if(st&4)
                  {
                        wcscat(buf_path, L"_lem.txt");
                        if(!_waccess(buf_path, 0)) _wremove(buf_path);
                        buf_path[bk]=L'\0';
```

```
                }
                if(st&8)
                {
                        wcscat(buf_path, L"_code.txt");
                        if(!_waccess(buf_path, 0)) _wremove(buf_path);
                }
        }
        if(st&1)
        {
                buf_path[bs+bl]=L'\0';
                bk=bs+bl+1;
                WideCharToMultiByte(CP_ACP, NULL, buf_path, bk,
command_11, bk, NULL, NULL);
                strcat(command, "index");
                system(command);
        }
    }
    if(st&9)
    {
        bs=wcslen(path);
        WideCharToMultiByte(CP_ACP, NULL, path, bs, command_11, bs,
NULL, NULL);
        command_11[bs]='\0';
        bs+=11;
        if(st&1)
        {
                strcat(command, "links_hash");
                system(command);
                command[bs]='\0';
                links_num=0;
        }
        if(st&8)
        {
                strcat(command, "sentence_hash");
                system(command);
                command[bs]='\0';
                strcat(command, "word_hash");
                system(command);
                wcscpy(buf_path, path);
                wcscat(buf_path, L"vocabulary.txt");
                if(!_waccess(buf_path, 0)) _wremove(buf_path);
                words=sentence_num=0;
        }
    }
}
void collection::remove()
{
    char command[70]="rmdir /s/q data\\collections\\";
    outmass(L"data\\collections.txt", collections, collections_len);
```

```
	strcat(command, name);
	name[0]='\0';
	system(command);
}
collection::~collection()
{
	unsigned int pos=collections_start,
		col_ins_len=0,
		i=0,
		nl=strlen(name);
	char buf[11],
		col_ins[100];
	if(!isexist())
	{
		delete [] collections;
		return;
	}
	for(i=0; i<nl||pos<collections_len; i++)
	{

	if(collections[pos+i+1]=='\"'||tolower(collections[pos+i+1])<tolo
wer(name[i]))
		{
			while(collections[pos]!='\n') pos++;
			pos++;
			i=0;
			continue;
		}
		if(tolower(collections[pos+i+1])>tolower(name[i])) break;
	}
	col_ins[col_ins_len++]='\"';
	for(i=0; i<nl; i++) col_ins[col_ins_len++]=name[i];
	col_ins[col_ins_len++]='\"';
	col_ins[col_ins_len++]=' ';
	toa(themes_num, buf);
	for(i=0; buf[i]!='\0'; i++) col_ins[col_ins_len++]=buf[i];
	col_ins[col_ins_len++]=' ';
	toa(docs_num, buf);
	for(i=0; buf[i]!='\0'; i++) col_ins[col_ins_len++]=buf[i];
	col_ins[col_ins_len++]=' ';
	toa(max_docs_theme, buf);
	for(i=0; buf[i]!='\0'; i++) col_ins[col_ins_len++]=buf[i];
	col_ins[col_ins_len++]=' ';
	toa(links_num, buf);
	for(i=0; buf[i]!='\0'; i++) col_ins[col_ins_len++]=buf[i];
	col_ins[col_ins_len++]=' ';
	toa(sentence_num, buf);
	for(i=0; buf[i]!='\0'; i++) col_ins[col_ins_len++]=buf[i];
	col_ins[col_ins_len++]=' ';
```

```
	toa(words, buf);
	for(i=0; buf[i]!='\0'; i++) col_ins[col_ins_len++]=buf[i];
	col_ins[col_ins_len++]='\r';
	col_ins[col_ins_len++]='\n';
	for(i=collections_len-1; i>=pos; i--)
collections[i+col_ins_len]=collections[i];
	collections_len+=col_ins_len;
	for(i=0; i<col_ins_len; i++) collections[pos+i]=col_ins[i];
	outmass(L"data\\collections.txt", collections, collections_len);
	if(themes_list!=NULL) delete [] themes_list;
	delete [] collections;
}

//Поиск оналогов документа
class analog{
	unsigned int freq_voc;
	wchar_t path[MAX_PATH];
public:
	analog(wchar_t *p, unsigned int f)
	{
		wcscpy(path, p);
		freq_voc=f;
	}

	void find_analog();
};
void analog::find_analog()
{
	unsigned int i,
		*mass,
		mass_len,
		word_val_len;
	double **word_val;
	wchar_t in_path[MAX_PATH],
		out_path[MAX_PATH];
	scaning *s1;
	lemma *l1;
	coding *c1;
	frequency_vocabulary *f1;
	wcscpy(out_path, path);
	wcscat(out_path, L"analog_out.txt");
	if(!_waccess(out_path, 0)) _wremove(out_path);
	wcscat(in_path, path);
	wcscpy(in_path, L"analog_in_scan.txt");
	if(!_waccess(in_path, 0)) _wremove(in_path);
	wcscat(in_path, path);
	wcscpy(in_path, L"analog_in_scan_lem.txt");
	if(!_waccess(in_path, 0)) _wremove(in_path);
	wcscat(in_path, path);
```

```
wcscpy(in_path, L"analog_in_scan_code.txt");
if(!_waccess(in_path, 0)) _wremove(in_path);
wcscpy(in_path, path);
wcscat(in_path, L"analog_in.txt");
if(_waccess(in_path, 0))return;
s1=new scaning(1, 1, 0, (unsigned char)152);
cout << "Сканирование..." << endl;
if(!s1->scan(in_path, "", 1)) return;
delete s1;
cout << "Лемматизация..." << endl;
l1=new lemma();
in_path[wcslen(in_path)-4]=L'\0';
wcscat(in_path, L"_scan.txt");
l1->initialize();
l1->lem(in_path);
delete l1;
cout << "Кодирование..." << endl;
wcscpy(in_path, path);
wcscat(in_path, L"word_hash\\properties");
ifstream in(in_path, ios::binary);
if(!in) return;
in.read((char*)&i, sizeof(i));
in.close();
c1=new coding(path, i, 0, 0, 0, 0, 0);
wcscpy(in_path, path);
wcscat(in_path, L"analog_in_scan_lem.txt");
c1->code_page(in_path, 0);
delete c1;
wcscpy(in_path, path);
wcscat(in_path, L"analog_in_scan_code.txt");
if(_waccess(in_path, 0)) return;
cout << "Составление спска слов и подсчёт функций..." << endl;
in.open(in_path, ios::binary);
if(!in) throw in_path;
in.seekg(0, ios::end);
mass_len=in.tellg()>>2;
mass=new unsigned int[mass_len];
in.seekg(0, ios::beg);
in.read((char*)mass, mass_len<<2);
in.close();
qsort(mass, mass_len, sizeof(int), reverse_compare);
f1=new frequency_vocabulary(path, freq_voc, 0);
word_val=new double*[mass_len];
word_val[0]=new double[6];
word_val[0][0]=mass[0];
word_val[0][2]=f1->get_word_freq((unsigned int)word_val[0][0]);
word_val[0][3]=f1->get_word_unic((unsigned int)word_val[0][0]);
for(i=1, word_val_len=1; mass[i]!=0; i++)
{
```

```
            if(mass[i-1]!=mass[i])
            {
                  word_val[word_val_len]=new double[6];
                  word_val[word_val_len][0]=mass[i];
                  word_val[word_val_len][1]=1;
                  word_val[word_val_len][2]=f1->get_word_freq((unsigned
int)word_val[word_val_len][0]);
                  word_val[word_val_len][3]=f1->get_word_unic((unsigned
int)word_val[word_val_len][0]);
                  word_val_len++;
            }
            else word_val[word_val_len-1][1]+=1;
      }
      delete [] mass;
      for(i=0; i<word_val_len; i++)
      {
            word_val[i][4]=f1->get_word_belong((unsigned
int)word_val[i][0], word_val, word_val_len);

      word_val[i][5]=(word_val[i][1]+word_val[i][1]*word_val[i][2]+word
_val[i][1]*word_val[i][4])*word_val[i][3]*word_val[i][3];
      }
      qsort(word_val, word_val_len, sizeof(*word_val),
word_val_compare);
      ofstream out(out_path, ios::binary);
      if(!out) throw out_path;
      for(i=0; i<40; i++)
      {
            out.write(f1->get_word((unsigned int)word_val[i][0]),
strlen(f1->get_word((unsigned int)word_val[i][0])));
            out.write(" ", 1);
      }
      out.write("\r\n\r\nСлово Частота Известность Уникальность
Принадлежность Релевантность\r\n", 73);
      for(i=0; i<word_val_len; i++)
      {
            out.write(f1->get_word((unsigned int)word_val[i][0]),
strlen(f1->get_word((unsigned int)word_val[i][0])));
            out.write(" ", 1);
            out << word_val[i][1] << " ";
            out << word_val[i][2] << " ";
            out << word_val[i][3] << " ";
            out << word_val[i][4] << " ";
            out << word_val[i][5] << " ";
            out.write("\r\n", 2);
            delete [] word_val[i];
      }
      delete [] word_val;
      out.close();
```

```
	delete f1;
}

int main(int argc, char* argv[])
{
	setlocale(LC_ALL, "Russian");
	try{
		unsigned int choise=0,
			themes_from,
			themes_to,
			docs_theme;
		char name[41]="\0",
			set[1024];
		collection *c1;
		while(true)
		{
			cout << "Введите команду:" << endl;
			cout << "1. Работать с коллекциями документов." <<
endl;
			cout << "2. Составить запрос, по которому можно найти
аналоги документа." << endl;
			cout << "3. Выход." << endl;
			cin >> choise;
			cout << endl;
			if(cin.fail())
			{
				cout << "Неверная команда.\n\n";
				cin.clear();
				cin.ignore(cin.rdbuf()->in_avail());
				continue;
			}
			switch(choise)
			{
			case 1:
				c1=new collection();
				while(true)
				{
					cout << "ВНИМАНИЕ: Не закрывайте программу
во время работы с коллекциями это вызовет их повреждение\n\n";
					cout << "Выберите коллекцию или создайте
новую.\n"
						<< "0. Создать новую коллекцию.\n";
					c1->print();
					cout << c1->get_collections_num()+1 << ".
Выход.\n";
					cin >> choise;
					cout << endl;
					if(cin.fail())
					{
```

```
                                    cout << "Неверная команда.\n\n";
                                    cin.clear();
                                    cin.ignore(cin.rdbuf() -> in_avail());
                                    continue;
                              }
                              if(choise==0)
                              {
                                    while(true)
                                    {
                                          cout << "Введите имя коллекции.
(латиница, кирилица, цифры и символы \"-\" \"_\" \" \"\n";
                                          cin.ignore(cin.rdbuf() ->
in_avail());
                                          SetConsoleCP(1251);
                                          cin.getline(name, sizeof(name));
                                          SetConsoleCP(866);
                                          cout << endl;
                                          if(cin.fail())
                                          {
                                                cout << "Имя коллекции
должно иметь длинну от 1 до 40 символов\n\n";
                                                cin.clear();
                                                cin.ignore(cin.rdbuf() ->
in_avail());
                                                continue;
                                          }
                                          try{ c1->create(name); }
                                          catch(wchar_t *ex)
                                          {
                                                wcout << L"\nОШИБКА:
Неудаётся создать файл \"" << ex << "\".\n" << endl;
                                                throw 1;
                                          }
                                          catch(char *ex)
                                          {
                                                cout << "Коллекция с именем
\"" << ex << "\" уже существует, либо в имени содержатся недопустимые
символы" << endl;
                                                continue;
                                          }
                                          break;
                                    }
                              }
                              else if(choise>0&&choise<=c1-
>get_collections_num()) c1->load(choise);
                              else if(choise==c1->get_collections_num()+1)
break;
                              else
                              {
```

```
                                    cout << "Нет такой команды.\n\n";
                                    continue;
                              }
                              while(true)
                              {
                                    cout << "1. Добавить документы по
списку (themes_list.txt)." << endl;
                                    cout << "2. Выполнить этапы обработки
коллекции." << endl;
                                    cout << "3. Определить колличество
слов с частотой большей или равной N." << endl;
                                    cout << "4. Посчитать частотный
словарь для коллекции." << endl;
                                    cout << "5. Настройки." << endl;
                                    cout << "6. Удалить коллекцию или
часть её данных." << endl;
                                    cout << "7. Выход." << endl;
                                    cin >> choise;
                                    cout << endl;
                                    if(cin.fail())
                                    {
                                          cout << "Неверная команда.\n\n";
                                          cin.clear();
                                          cin.ignore(cin.rdbuf()-
>in_avail());
                                          continue;
                                    }
                                    switch(choise)
                                    {
                                    case 1:
                                          c1->prepare_themes_list();
                                          if(c1->get_themes_list_num()==0)
                                          {
                                                cout << "В списке не
найдено ни одной темы.\n\n";
                                                continue;
                                          }
                                          cout << "В списке найдено: " <<
c1->get_themes_list_num() << " тем.\n"
                                                << "Введите с какой по
какую тему использовать для добавления документов.\n"
                                                << "Если включена настройка
\"use_theme_hash\" то для уже имеющимся в коллекции тем не создаются
папки, а документы добавляются только если их до этого было собрано
меньше чем нужно теперь.\n";
                                          while(true)
                                          {
                                                while(true)
                                                {
```

```
                                                cout << "Введите
начальную позицию:" << endl;
                                                cin >> themes_from;
                                                cout << endl;
                                                if(cin.fail())
                                                {
                                                      cout << "Неверная
команда.\n\n";
                                                      cin.clear();

      cin.ignore(cin.rdbuf()->in_avail());
                                                      continue;
                                                }

      if(themes_from>=0&&themes_from<=c1->get_themes_list_num()) break;
                                                else cout << "Неверное
значение.\n\n";
                                          }
                                          while(true)
                                          {
                                                cout << "Введите
конечную позицию." << endl;
                                                cin >> themes_to;
                                                cout << endl;
                                                if(cin.fail())
                                                {
                                                      cout << "Неверная
команда.\n\n";
                                                      cin.clear();

      cin.ignore(cin.rdbuf()->in_avail());
                                                      continue;
                                                }

      if(themes_to>0&&themes_to<=c1->get_themes_list_num()) break;
                                                else cout << "Неверное
значение.\n\n";
                                          }
                                          cout << "Будут использованы
темы с " << themes_from << " по " << themes_to << " всё верно?" <<
endl;
                                          cout << "1. Да" << endl;
                                          cout << "2. Нет" << endl;
                                          cin >> choise;
                                          cout << endl;
                                          if(cin.fail())
                                          {
                                                cout << "Неверная
команда.\n\n";
```

```
                                                cin.clear();
      cin.ignore(cin.rdbuf()->in_avail());
                                                continue;
                                          }
                                          switch(choise)
                                          {
                                          case 1:
                                                break;
                                          case 2:
                                                continue;
                                          default:
                                                cout << "Неверная
команда.\n" << endl;
                                                continue;
                                          }
                                          break;
                                    }
                                    cout << "Введите количество
документов которые нужно скачать по каждой теме.\n";
                                    cout << "\n(0 = выход)\n";
                                    while(true)
                                    {
                                          cin >> docs_theme;
                                          cout << endl;
                                          if(cin.fail())
                                          {
                                                cout << "Неверная
команда.\n\n";
                                                cin.clear();
      cin.ignore(cin.rdbuf()->in_avail());
                                                continue;
                                          }
                                          if(docs_theme>=0) break;
                                          else cout << "Неверное
значение\n\n";
                                    }
                                    if(docs_theme==0) continue;
                                    try
                                    {
                                          c1->add_list(themes_from,
themes_to, docs_theme);
                                    }
                                    catch(unsigned int ex)
                                    {
                                          cout << "\nОбъём хэш таблиц
превысил выделенную для него память (" << ex << ").\n" << endl;
                                    }
```

```
                                        continue;
                                    case 2:
                                        while(true)
                                        {
                                            cout << "Чтобы выполнить
этап введите:\n"
                                            << "1. Выделение
предложений.\n"
                                            << "2. Лемматизация.\n"
                                            << "4. Кодирование.\n"
                                            << "Или введите их сумму,
для выполнения нескольких этапов сразу.\n"
                                            << "ВНИМАНИЕ! При
выполнении действуют все настройки, относящиеся к этапу, кроме
настройки каторая его включает.\n"
                                            << "Если провести
кодирование с включенной проверкой предложений на плагиат перед этим
не удалив данные этого этапа, то получится, что все документы состоят
полностью из плагиата.\n"
                                            << "Введита 0 для
отмены.\n";
                                            cin >> choise;
                                            cout << endl;
                                            if(cin.fail())
                                            {
                                                cout << "Неверная
команда.\n\n";
                                                cin.clear();
    cin.ignore(cin.rdbuf()->in_avail());
                                                continue;
                                            }
                                            if(!choise) break;
                                            if(choise>7)
                                            {
                                                cout << "Нет такой
команды.\n\n";
                                                continue;
                                            }
                                            c1->make_stages(choise);
                                            break;
                                        }
                                        continue;
                                    case 3:
                                        while(true)
                                        {
                                            cout << "Введите N." <<
endl;
                                            cin >> choise;
```

```
                                        cout << endl;
                                        if(cin.fail())
                                        {
                                              cout << "Неверная
команда.\n\n";
                                              cin.clear();

      cin.ignore(cin.rdbuf()->in_avail());
                                              continue;
                                        }
                                        cout << "В коллекции " <<
c1->gt_frequency(choise) << " слов с частотой большей или равной " <<
choise << ".\n" << endl;
                                        break;
                                  }
                                  continue;
                            case 4:
                                  while(true)
                                  {
                                        char choise;
                                        cout << "Подсчёт частот
может занять очень продолжительное время. Вы уверены, что хотите
запустить его?\n"
                                              << "Y. да.\n"
                                              << "N. нет.\n";
                                        cin >> choise;
                                        cout << endl;
                                        if(cin.fail())
                                        {
                                              cout << "Неверная
команда.\n\n";
                                              cin.clear();

      cin.ignore(cin.rdbuf()->in_avail());
                                              continue;
                                        }

      if(choise=='Y'||choise=='y')
                                        {
                                              cout << "Не закрывайте
программу пока она не завершит работу.\n" << endl;
                                              c1->freq_calculate();
                                        }

      if(choise!='N'&&choise!='n'&&choise!='Y'&&choise!='y')
                                        {
                                              cout << "Нет такой
командаы.\n" << endl;
                                              continue;
```

```
                                        }
                                        break;
                                    }
                                    continue;
                                case 5:
                                    while(true)
                                    {
                                        cout << "1. Использовать
хэш для тем = " << c1->use_theme_hash.get() << endl;
                                        cout << "2. Длинна хэша для
тем = " << c1->theme_row_size.get() << endl;
                                        cout << "3. Использовать
Яндекс XML = " << c1->yandex_xml.get() << endl;
                                        cout << "4. Опции запроса,
для Яндекс XML = " << c1->yandex_xml_options.get() << endl;
                                        cout << "5. Точный запрос =
" << c1->exact_query.get() << endl;
                                        cout << "6. Сохранять
ссылки = " << c1->save_links.get() << endl;
                                        cout << "7. Использовать
список исключаемых серверов = " << c1->use_server_except.get() <<
endl;
                                        cout << "8. Использовать
хэш использованных ссылок = " << c1->use_links_hash.get() << endl;
                                        cout << "9. Длинна хэша для
ссылок = " << c1->links_row_size.get() << endl;
                                        cout << "10. Удалять
страницы оглавления Яндекса = " << c1->delete_index.get() << endl;
                                        cout << "11. Удалять
скачанные документы после сканирования предложений = " << c1-
>delete_page.get() << endl;
                                        cout << "12. Сканировать
документы выделяя предложения = " << c1->use_sentence_scan.get() <<
endl;
                                        cout << "13. Минимальная
длинна предложения = " << c1->min_length.get() << endl;
                                        cout << "14. Минимальное
колличество слов в пердложении = " << c1->min_words.get() << endl;
                                        cout << "15. Сохранять
открывающие теги (абзацы) = " << c1->save_open_tag.get() << endl;
                                        cout << "16. Минимальное
колличество предложений в документе = " << c1->min_sentences.get() <<
endl;
                                        cout << "17. Удалять
документы с результатами сканирования после лемматизации = " << c1-
>delete_scan_page.get() << endl;
                                        cout << "18. Использовать
лемматизацию = " << c1->use_lemmatization.get() << endl;
```

```
                                        cout << "19. Удалять
лемматизованные документы после кодирования = " << c1-
>delete_lemma_page.get() << endl;
                                        cout << "20. Кодировать
документы = " << c1->use_word_coding.get() << endl;
                                        cout << "21. Длинна хэша
для словаря = " << c1->word_row_size.get() << endl;
                                        cout << "22. Сортировать
слова в предложении при кодировании = " << c1->use_sentence_sort.get()
<< endl;
                                        cout << "23. Проверять
предложения на плагиат = " << c1->use_sentence_hash.get() << endl;
                                        cout << "24. Длинна хэша
для предложений = " << c1->sentence_row_size.get() << endl;
                                        cout << "25. Максимально
возможный объём памяти выделенный под хэш таблицы = " << c1-
>max_all_hash_size.get() << endl;
                                        cout << "26. Символы из
utf8 отсутствующие в cp1251 заменять на символ с номером = " << c1-
>missing_symbol.get() << endl;
                                        cout << "27. Минимальная
частота слова для попадания в частотный словарь = " << c1-
>min_frequency.get() << endl;
                                        cout << "28. Память для
кэша частотного словаря = " << c1->freq_vocabulary_mem.get() << endl;
                                        cout << "29. По умолчанию"
<< endl;
                                        cout << "30. Выход" <<
endl;

                                        cin >> choise;
                                        cout << endl;
                                        if(cin.fail())
                                        {
                                              cout << "Неверная
команда.\n\n";
                                              cin.clear();

      cin.ignore(cin.rdbuf()->in_avail());
                                              continue;
                                        }
                                        switch(choise)
                                        {
                                        case 1:
                                              while(true)
                                              {
                                                    cout <<
"Проверять существует ли тема в коллекции (используется хэш)." <<
endl;
```

```
                                                cout << "Если не
использовать проверку, то любая тема в файле themes_list.txt будет
считаться новой и по ней будут добавлены документы." << endl;
                                                cout << "В
последствии исключить совпадающие темы будет невозможно." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>use_theme_hash.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 2:
                                          while(true)
                                          {
                                                cout << "Введите
длинну хэша для тем." << endl;
                                                cout <<
"Оптимальная длинна это простое число в полтора раза большее
предполагаемого колличества тем в коллекции." << endl;
                                                cout << "Так же
можно использовать не простое число а (2^n)-1." << endl;
                                                cout << "Если
равно \"0\" то длинна определяется автоматически при каждом запуске
добавления документов. Желательно задать длинну вручную так как при её
изменении придётся пересоздавать хэш, что дольше чем загружать его."
<< endl;
                                                cin >> set;
                                                try{c1-
>theme_row_size.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 3:
```

```
                                                while(true)
                                                {
                                                      cout <<
"Использовать вместо обычного Яндекс поиска сервис яндекс XML
(необходим аккаунт в данном сервисе и лимит запросов)." << endl;
                                                      cout << "0. Нет."
<< endl;
                                                      cout << "1. Да."
<< endl;
                                                      cin >> set;
                                                      try{c1-
>yandex_xml.set(set);}
                                                      catch(char *ex)
                                                      {
                                                            cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                            continue;
                                                      }
                                                      break;
                                                }
                                                continue;
                                          case 4:
                                                while(true)
                                                {
                                                      cout << "Введите
строку параметров запроса для Яндекс XML (см. \"Помощь\" в Яндекс
XML). Параметры query и page задаются программой. Параметры должны
начинаться с \"&\"." << endl;
                                                      cout << "Если не
хотите ничего менять введите \"0\"." << endl;
                                                      cin >> set;
                                                      if(*set=='0')
break;
                                                      c1-
>yandex_xml_options.set(set);
                                                      break;
                                                }
                                                continue;
                                          case 5:
                                                while(true)
                                                {
                                                      cout <<
"Использовать при поиске в Яндексе именно то, что в теме (без
добавления результатов и исправления ошибок)." << endl;
                                                      cout << "0. Нет."
<< endl;
                                                      cout << "1. Да."
<< endl;
                                                      cin >> set;
```

```
                                                try{c1-
>exact_query.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 6:
                                          while(true)
                                          {
                                                cout <<
"Сохранять ссылку по которой скачан документ." << endl;
                                                cout <<
"Записывается в начале документа между тегами <selflink> и
</selflink>." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>save_links.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 7:
                                          while(true)
                                          {
                                                cout << "Не
скачивать документы с серверов из списка исключений (используется
хэш)." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>use_server_except.set(set);}
                                                catch(char *ex)
```

```
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 8:
                                          while(true)
                                          {
                                                cout << "Не
скачивать документы по уже использованным ссылкам. (используется
хэш)." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>use_links_hash.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 9:
                                          while(true)
                                          {
                                                cout << "Введите
длинну хэша для ссылок." << endl;
                                                cout <<
"Оптимальная длинна это простое число в полтора раза большее
предполагаемого колличества ссылок использованных при сборе
коллекции." << endl;
                                                cout << "Так же
можно использовать не простое число а (2^n)-1." << endl;
                                                cout << "Если
равно \"0\" то длинна определяется автоматически при каждом запуске
добавления документов. Желательно задать длинну вручную так как при её
изменении придётся пересоздавать хэш, что дольше чем загружать его."
<< endl;
                                                cin >> set;
                                                try{c1-
>links_row_size.set(set);}
```

```
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 10:
                                          while(true)
                                          {
                                                cout << "Удалять
страницы Яндекса, которые использовались для поиска ссылок по теме."
<< endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>delete_index.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 11:
                                          while(true)
                                          {
                                                cout << "Удалять
html страницы после того как из них выделены предложения. Работает
только при включённом сканировании." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>delete_page.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
```

```
                                                  }
                                                  break;
                                            }
                                            continue;
                                      case 12:
                                            while(true)
                                            {
                                                  cout <<
"Сканировать документы удаляя html код и всё что не является
предложением." << endl;
                                                  cout <<
"Предложением является текст заканчивающийся на точку восклицательный
или вопросительный знак за которым следует пробел или тег. Также
предложению должен предшествовать открывающий тег и оно не должно
прерываться закрывающими тегами." << endl;
                                                  cout << "0. Нет."
<< endl;
                                                  cout << "1. Да."
<< endl;
                                                  cin >> set;
                                                  try{c1-
>use_sentence_scan.set(set);}
                                                  catch(char *ex)
                                                  {
                                                        cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                        continue;
                                                  }
                                                  break;
                                            }
                                            continue;
                                      case 13:
                                            while(true)
                                            {
                                                  cout << "Введите
минимальное колличество символов в предложении. Предложения с меньшим
колличеством символов не будут сохраняться." << endl;
                                                  cin >> set;
                                                  try{c1-
>min_length.set(set);}
                                                  catch(char *ex)
                                                  {
                                                        cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                        continue;
                                                  }
                                                  break;
                                            }
                                            continue;
```

```
                                        case 14:
                                            while(true)
                                            {
                                                cout << "Введите
минимальное колличество слов в предложении. Предложения с меньшим
колличеством слов не будут сохраняться." << endl;
                                                cin >> set;
                                                try{c1-
>min_words.set(set);}
                                                catch(char *ex)
                                                {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
                                                }
                                                break;
                                            }
                                            continue;
                                        case 15:
                                            while(true)
                                            {
                                                cout <<
"Сохранять открывающий тег. Он выводиться перед группой предложений и
обозначает, что они находятся в одном абзаце." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>save_open_tag.set(set);}
                                                catch(char *ex)
                                                {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
                                                }
                                                break;
                                            }
                                            continue;
                                        case 16:
                                            while(true)
                                            {
                                                cout << "Введите
минимальное колличество предложений в документе. Докименты с меньшим
колличеством предложений не будут добавляться в коллекцию." << endl;
                                                cin >> set;
                                                try{c1-
>min_sentences.set(set);}
```

```
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 17:
                                          while(true)
                                          {
                                                cout << "Удалять
результаты сканирования после того как они лемматизированы. Работает
только при включенной лемматизации." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>delete_scan_page.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 18:
                                          while(true)
                                          {
                                                cout <<
"Лемматизировать документы. Настройка не будет работать если не
включено сканирование предложений." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>use_lemmatization.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
```

```
                                                }
                                                break;
                                          }
                                          continue;
                                    case 19:
                                          while(true)
                                          {
                                                cout << "Удалять
лемматизированные документы после кодирования." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>delete_lemma_page.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 20:
                                          while(true)
                                          {
                                                cout <<
"Составить пронумерованный словарь и кодировать документы по нему
(используется хэш)." << endl;
                                                cout << "Не
работает если выключена лемматизация." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>use_word_coding.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
```

```
                                        case 21:
                                            while(true)
                                            {
                                                cout << "Введите
длинну хэша для слов." << endl;
                                                cout <<
"Оптимальная длинна это простое число в полтора раза большее
предполагаемого колличества слов в словаре." << endl;
                                                cout << "Так же
можно использовать не простое число а (2^n)-1." << endl;
                                                cout << "Если
равно \"0\" то длинна определяется автоматически при каждом запуске
добавления документов. Желательно задать длинну вручную так как при её
изменении придётся пересоздавать хэш, что дольше чем загружать его."
<< endl;
                                                cin >> set;
                                                try{c1-
>word_row_size.set(set);}
                                                catch(char *ex)
                                                {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
                                                }
                                                break;
                                            }
                                            continue;
                                        case 22:
                                            while(true)
                                            {
                                                cout << "Нужно ли
сортировать слова в предложении по возрастанию их номеров? (позволяет
определять плагиатные предложения с изменением порядка слов)" << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>use_sentence_sort.set(set);}
                                                catch(char *ex)
                                                {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
                                                }
                                                break;
                                            }
                                            continue;
```

```
                                        case 23:
                                            while(true)
                                            {
                                                cout <<
"Проверять предложения на плагиат. (используется хэш)." << endl;
                                                cout << "0. Нет."
<< endl;
                                                cout << "1. Да."
<< endl;
                                                cin >> set;
                                                try{c1-
>use_sentence_hash.set(set);}
                                                catch(char *ex)
                                                {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
                                                }
                                                break;
                                            }
                                            continue;
                                        case 24:
                                            while(true)
                                            {
                                                cout << "Введите
длинну хэша для предложений." << endl;
                                                cout <<
"Оптимальная длинна это простое число в полтора раза большее
предполагаемого колличества предложений в коллекции." << endl;
                                                cout << "Так же
можно использовать не простое число а (2^n)-1." << endl;
                                                cout << "Если
равно \"0\" то длинна определяется автоматически при каждом запуске
добавления документов. Желательно задать длинну вручную так как при её
изменении придётся пересоздавать хэш, что дольше чем загружать его."
<< endl;
                                                cin >> set;
                                                try{c1-
>sentence_row_size.set(set);}
                                                catch(char *ex)
                                                {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
                                                }
                                                break;
                                            }
                                            continue;
                                        case 25:
```

```
                                        while(true)
                                        {
                                              cout << "Введите
какой объём оперативной памяти в байтах могут занимать все хэш
таблицы." << endl;
                                              cin >> set;
                                              try{c1-
>max_all_hash_size.set(set);}
                                              catch(char *ex)
                                              {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
                                              }
                                              break;
                                        }
                                        continue;
                                  case 26:
                                        while(true)
                                        {
                                              cout << "Введите
номер символа в кодировке cp1251 (0-255) на который будут заменены
символы из utf8, которых нет в cp1251." << endl;
                                              cin >> set;
                                              try{c1-
>missing_symbol.set(set);}
                                              catch(char *ex)
                                              {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
                                              }
                                              break;
                                        }
                                        continue;
                                  case 27:
                                        while(true)
                                        {
                                              cout << "Введите
Минимальную частоту встречи слова начиная с которай оно попадёт в
таблицу частотного словаря." << endl;
                                              cin >> set;
                                              try{c1-
>min_frequency.set(set);}
                                              catch(char *ex)
                                              {
                                                    cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                    continue;
```

```
                                                }
                                                break;
                                          }
                                          continue;
                                    case 28:
                                          while(true)
                                          {
                                                cout << "Введите
количество памяти в байтах которую сможет использовать кэш для
частотного словаря." << endl;
                                                cin >> set;
                                                try{c1-
>freq_vocabulary_mem.set(set);}
                                                catch(char *ex)
                                                {
                                                      cout <<
"Значение \"" << ex << "\" не подходит." << endl;
                                                      continue;
                                                }
                                                break;
                                          }
                                          continue;
                                    case 29:
                                          c1-
>col_settings.def();
                                          continue;
                                    case 30:
                                          c1-
>col_settings.save();
                                          break;
                                    default:
                                          cout << "Нет такой
коменды\n\n";
                                          continue;
                                    }
                                    break;
                              }
                              continue;
                        case 6:
                              while(true)
                              {
                                    cout << "Чтобы удалить
коллекцию введите 16.\n"
                                          << "Чтобы удалить
данные этапа введите:\n"
                                          << "1. Скачанные
страницы, оглавление, хеш использованнх ссылок.\n"
                                          << "2. Файлы с
выделенными предложениями.\n"
```

```
                                                << "4.
Лемматизированные файлы.\n"
                                                << "8. Кодированные
файлы, словарь, хеш слов,  хеш предложений.\n"
                                                << "Или их сумму, для
удаления нескольких этапов сразу.\n"
                                                << "Введите 0 для
отмены.\n";
                                          cin >> choise;
                                          cout << "\n\n";
                                          if(cin.fail())
                                          {
                                                cout << "Неверная
команда.\n\n";
                                                cin.clear();

      cin.ignore(cin.rdbuf()->in_avail());
                                                continue;
                                          }
                                          if(!choise) break;
                                          else if(choise>16)
                                          {
                                                cout << "Неверная
команда.\n\n";
                                                continue;
                                          }
                                          while(true)
                                          {
                                                char ch;
                                                cout << "Вы
действительно хотите удалить";
                                                if(choise&16) cout <<
" всю коллекцию?\n";
                                                else
                                                {
                                                      cout << ":\n";
                                                      if(choise&1) cout
<< "- Скачанные страницы, оглавление, хеш использованнх ссылок.\n";
                                                      if(choise&2) cout
<< "- Файлы с выделенными предложениями.\n";
                                                      if(choise&4) cout
<< "- Лемматизированные файлы.\n";
                                                      if(choise&8) cout
<< "- Кодированные файлы, словарь, хеш слов,  хеш предложений.\n";
                                                }
                                                cout << "Y. Да.\n"
                                                      << "N. Нет.\n";
                                                cin >> ch;
                                                cout << endl;
```

```
                                                if(cin.fail())
                                                {
                                                      cout << "Неверная
команда.\n\n";
                                                      cin.clear();
      cin.ignore(cin.rdbuf()->in_avail());
                                                      continue;
                                                }
                                                if(ch=='Y'||ch=='y')
                                                {
                                                      if(choise==16)
c1->remove();
                                                      else c1-
>remove_stages(choise);
                                                      cout << "Удаление
завершено\n\n";
                                                      break;
                                                }
                                                else
if(ch=='N'||ch=='n') break;
                                                      else cout << "Нет
такой команды\n\n";
                                          }
                                          if(c1->isexist()) continue;
                                          break;
                                    }
                                    if(c1->isexist()) continue;
                                    break;
                              case 7:
                                    cout << "Подождите сохранение
данных" << endl;
                                    break;
                              default:
                                    cout << "Нет такой коменды\n\n";
                                    continue;
                              }
                              break;
                        }
                        break;
                  }
                  delete c1;
                  break;
            case 2:
                  while(true)
                  {
                        c1=new collection();
                        cout << "Выберите коллекцию данные которой
будут использованы при поиске аналогов." << endl;
```

```
                              cout << "ВНИМАНИЕ! Не запускайте поиск
аналогов если вы выполняли кодирование документов коллекции и не
выполнили пересчёт частотного словаря." << endl;
                              cout << "Если вы изменили значение частоты
при которой слово попадает в словарь верните значение назад или
выполните пересчёт частотного словаря с этим значением." << endl;
                              cout << "Внутри коллекции должен быть создан
файл analog_in.txt и в него помещён текст по которому нужно найти
аналоги." << endl;
                              cout << "0. Выход." << endl;
                              c1->print();
                              cin >> choise;
                              cout << endl;
                              if(cin.fail())
                              {
                                    cout << "Неверная команда.\n\n";
                                    cin.clear();
                                    cin.ignore(cin.rdbuf() -> in_avail());
                                    continue;
                              }
                              if(choise==0) break;
                              else if(choise>0&&choise<=c1-
>get_collections_num()) c1->load(choise);
                              else
                              {
                                    cout << "Нет такой команды.\n\n";
                                    continue;
                              }
                              analog a1(c1->get_path(), c1-
>min_frequency.get());
                              delete c1;
                              a1.find_analog();
                              cout << "Запрос по которому можно найти
аналоги, а так же данные по словам сохранены в файле analog_out.txt\n"
<< endl;
                              break;
                        }
                        break;
                  case 3:
                        return 0;
                  default:
                        cout << "Нет такой команды.\n\n";
                        continue;
                  }
            }
      }
      catch(int ex)
      {
            cout << "Программа завершена с кодом ошибки "<< ex << endl;
```

```
	}
	catch(wchar_t ex)
	{
		wcout << L"ОШИБКА: Неудаётся создать папку \"" << ex <<
"\"." << endl;
	}
	return 0;
}
```